MCR-70-424

DEVELOPMENT OF A TEST AND FLIGHT ENGINEERING ORIENTED LANGUAGE

PHASE III REPORT

W. F. Kamsler
C. W. Case
E. L. Kinney
J. Gyure

Martin Marietta Corporation
Denver Division
P. O. Box 179
Denver, Colorado 80201

December, 1970

Phase III Report for October - December 1970

Prepared for
National Aeronautics and Space Administration
John F. Kennedy Space Center

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
MCR-70-424		

4. Title and Subtitle	5. Report Date
Development of a Test And Flight Engineering Oriented Language Phase III Report	December 1970
	6. Performing Organization Code
7. Author(s) W. F. Kamsler, C. W. Case, J. Gyure, E. L. Kinney	8. Performing Organization Report No. MCR-70-424
9. Performing Organization Name and Address MARTIN MARIETTA CORPORATION Denver Division P. O. Box 179 Denver, Colorado 80201	10. Work Unit No.
	11. Contract or Grant No. NAS10-7308
12. Sponsoring Agency Name and Address National Aeronautics and Space Adminstration Kennedy Space Center Florida 32899	13. Type of Report and Period Covered Phase III Report October 1970 to December 1970
	14. Sponsoring Agency Code

15. Supplementary Notes

Phase I Report = MCR-70-327, Martin Marietta, Denver
Phase II Report = MCR-70-365, Martin Marietta, Denver

16. Abstract

This report covers Phase III activity in the development of a test and flight engineering language.

Based on an analysis of previously developed test oriented languages (Phase I) and a study of test language requirements (Phase II) a high order language has been designed to enable test and flight engineers to checkout and operate the proposed Space Shuttle and other NASA vehicles and experiments.

The language is called ALOFT:

A Language Oriented to Flight Engineering and Testing

The report describes the language, compares its terminology to similar terms in other test languages, and discusses its features and utilization.

The appendix provides the specifications for ALOFT.

17. Key Words	18. Distribution Statement
Test Oriented Language Space Shuttle Computer Controlled Test Equipment ALOFT	

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
UNCLASSIFIED	UNCLASSIFIED	95	

CONTENTS

1.0 INTRODUCTION

This report covers Phase III of the study task for development of a Test and Flight Engineer Oriented Language.

From the analysis of previously developed languages studied in Phase I and from the needed language characteristics determined in Phase II, a high-order language has been designed to enable test and flight engineers to check out and operate the proposed Space Shuttle and other NASA vehicles and experiments. This language has been named A L O F T - A Language Oriented to Flight Engineering and Testing.

The language has been designed to be functionally independent of any test system and be flexible enough for use with a wide variety of future space vehicles and experiments. Test system independence is of extreme importance when a language is developed prior to the design of the test system. Making the language flexible enough to handle Space Shuttle tasks, and yet not restricting it to just that activity, permits use of the language in various test systems.

Prime importance has been given to the readability of the language to ensure its acceptance. Writing aids will assist the writer in preparing his tests. The English-like syntactic structure of the language makes it easy to write. The language readily adapts to all technical disciplines.

Through the use of the ALOFT language, checkout, test, and operating procedures can be efficiently prepared and performed.

The specification for ALOFT is presented in the Appendix.

2.0 SPACE SHUTTLE ORIENTED USAGE

ALOFT, the language oriented to test and flight engineering is being designed to be independent of the test system and of Space Shuttle itself. The language does, however, contain features to enable it to cope with Space Shuttle peculiar features and requirements.

2.1 Characteristics of ALOFT

The study of the Space Shuttle performed in Phase II* determined that the capabilities described in the following paragraphs should be included in the language.

Test oriented capabilities for:

- Test initiation;
- Application of stimulus;
- Measurement of output;
- Comparison of results;
- Man/machine interfaces;
- Records and logs with time tags;
- Monitoring;
- Clock and time controlled actions;
- System, subsystem, and unit testing.

Independence with respect to testing equipment via:

- Dictionary data banks;
- Common character set;
- Free form with respect to input media;
- No interaction with operating system;
- Test writer-created safing features.

*Martin Marietta Report MCR-70-365.

Flexibility provided by:

- Full arithmetic and relational operator set;
- Thirty-two character data names;
- List and table capability;
- Simple loop capability;
- Subroutines;
- Integer, fixed point, Boolean, text, binary, and time data;
- Simple numeric and Boolean assignment statements;
- Unconditional and simple conditional transfers;
- Interrupt initiated routines.

Engineering reader orientation with:

- English words for primitives;
- Natural English forms as delimiters;
- Natural statement structure;
- Comments are easily accommodated.

Concurrent test execution provisions:

- Initiated via language primitives;
- Synchronization capability;
- Interrupt capability;
- Meaning dependent on language processor implementation.

Self-extension through:

- Macro definition capability;
- Other language capability;
- Programmer ability to create new primitives from existing core set and create specialized subroutines in other languages.

Special communications requirements:

- Computer to computer;
- Computer to data bus.

2.2 Special Space Shuttle Requirements

Within the capabilities of ALOFT are special Space Shuttle unique capabilities. These capabilities are included in the language in such a manner that does not preclude use of the language to test and operate other NASA space vehicles and experiments.

2.2.1 Multi-Discipline Terminology - The Space Shuttle encompasses many disciplines, each with its own terminology. To facilitate writing and reading test programs, a very flexible, yet unambiguous language structure is provided. A minimum number of rules and restraints are imposed on the user.

A basic English-like statement structure is used for test action statements. It has the form:

When - do what - to what.

The *when* permits a time statement to define when the desired action is to take place. This can be at a specific time, every so many seconds, etc.

The *do what* defines the action that is to take place. To meet the specific needs of the many Space Shuttle disciplines, a variety of action words are necessary. The language provides this capability. Typical are such verbs as measure, verify, apply, set, turn, send, display, print, etc. With this variety of verbs the user is able to select terms that most accurately describe the action. For example, he might desire to *set* a value *open*. To require him to *apply* a valve *on* would be extremely awkward, and the readability of the language would suffer. Chapter 3 provides the reasons for selection of language terminology.

The *to what* phrase identifies the name of the function undergoing the action. The *name* is defined in the dictionary data bank. For any given test program and test system these *names* are defined in terms that are meaningful in relation to the article under test. Provisions are included to enable these functions (which will appear in signal lists, schematics, etc) to be so defined.

These defined *names* are placed in the dictionary using SPECIFY statements. The test writer is confined to the use of function *names* which must eventually appear in the dictionary.

The dictionary also facilitates the problem of identifying the calling addresses of Space Shuttle systems, subsystems, LRUs etc. The redundant data bus concept of the Space Shuttle requires all

addressable items to be identified by their data bus and interface unit (IU) numbers. The data bus, IU, and function codes are identified at the same time the function *name* is placed in the data dictionary.

2.2.2 Language Concept - Typical examples of the language, ready for compiling are:

```
STATEMENT 80 AFTER CDC ''COUNT DOWN CLOCK'' IS -10MIN 50SEC,
        MEASURE_ RIGHT AILERON 2 POSITION_ AND SAVE AS _
        AILERON POSITION_ .
        IF_ AILERON POSITION_ IS BETWEEN 10PCT AND 20PCT GO
        TO STATEMENT 120.
                    ●
                    ●
                    ●
                    ●
STATEMENT 120 DISPLAY TEXT (RIGHT AILERON 2 IN CORRECT POSITION)
          ON_ CRT 2, LINE 23_ .
```

As is readily seen, the language is very readable. All specially defined items are delimited by means of underscores. The compiler obtains the address for such information from the dictionary.

Comments such as "countdown clock," which are not to be compiled, are delimited by dual apostrophes. (Quotation marks are not available on most digital printers.)

Text to be printed or displayed such as "Right Aileron 2 in correct position," is delimited by open and closed parentheses.

If the dictionary (as proposed) were not provided, it would become necessary for the test writer to define addresses while writing the test procedure. The procedure would then be more subject to error and the printed address data would impair readability.

2.2.3 Dimensional Capability - To further facilitate use of the language, a large vocabulary of dimensional terms and units is provided. It is possible to dimension a current measurement as either 0.003A or 3MA. This permits fixed point arithmetic, with the multiplier or exponent explicit in the dimensional term.

2.2.4 Concurrent Programs and Continuous Monitoring - A requirement stated for the language when applied to the Space Shuttle is that it be able to specify and control programs which must run concurrently. ALOFT provides this capability by prefixing a PERFORM PROGRAM statement with the word CONCURRENTLY. When the CONCURRENTLY PERFORM PROGRAM *name* is encountered, the operating systems will cause both programs to operate together.

A similar requirement for the language is that it be able to call for the recurrent verification or monitoring of a group of functions while a test is in progress. Such a feature is necessary to ensure that related support functions are operating correctly during the test and/or to verify that the test in progress is not causing any unexpected interactions in other parts of the system.

This requirement is satisfied by the provision of a repetitive phrase EVERY *time interval* prefixed to a VERIFY statement. When this prefix is used, the main program continues but is instantaneously interrupted at the specified time intervals to repeat the VERIFY statement. The repeated verifications or monitoring will continue until a RELEASE statement is encountered in the program.

2.2.5 Table Construction - To facilitate monitoring and other checkout and test functions, tables and lists can be prepared to minimize the writing task of the user. Such a table could contain various function names, units, their limits, and the default action to be taken if a function does not satisfy the test. The tables can also be used to store successive value readings of the functions to provide a profile.

2.2.6 Safing - The test writer can construct safing routines to provide "backout" and "return to original condition" programs. These programs can be called by any default condition. To ensure that they are not interrupted, these subroutines are prefixed with BEGIN CRITICAL_ _ _ _ .

2.2.7 Flexibility - While specifically ensuring that the language will meet Space Shuttle requirements, the use of the language to test other systems has not been overlooked. For example, the language can be used with most (if not all) existing real-time computer controlled systems to accomplish the preparation of programs for checkout, data acquisition, telemetry, control, etc.

3.0 LANGUAGE TERMINOLOGY COMPARISONS

The terminology used in existing test oriented languages (TOLs) has been evaluated for possible application to ALOFT. Table 1 introduces the operators identified during the analysis for comparison purposes. The language elements included in ATOLL, ATLAS, and CLASP were emphasized for the following reasons: ATOLL is representative of the most used TOL; ATLAS is the most recent TOL developed and uses the most English-like statements; CLASP, because of its potential use as an airborne computer language with which ALOFT may likely interface. CLASP elements found to be of use in a test language appear under the column-heading "OTHER." The test-action items that appear in the left-hand column were selected as representative of the ten languages studied. The operators that appear next to a test-action statement do not necessarily fully comply with the action. In some instances, the operator represents a close approximation or partial fulfillment of the test action. This was necessary to keep the table from increasing to an unwieldy size.

The basic requirements to be satisfied by a TOL are directed toward the test actions necessary to conduct a test. These operations involve the application of a stimulus, acquisition of the resulting output, comparison of the acquired value with predetermined limits, and a decision made as result of the comparison. Controls associated with the stimulus generating device necessitate selection and connections between the test system and the Unit Under Test (UUT) must be made. Controls associated with the measurement device require selection; and connections have to be made to route the test signal to the measurement unit. The resulting measured value will be compared with predetermined limits to determine its acceptability. A decision must be made which is dependent upon the acceptability of the measured value: continue with the present test sequence or branch to a malfunction isolation or termination routine.

Test systems exist that provide much less than the capabilities described above. A passive monitoring system is an example of such a system. In contrast, there are test systems that exceed the capabilities described. With the exception of ATLAS, TOLs developed to this date have been designed for a specific test program using identifiable test equipment. The languages are test system/test article dependent. Modification of such a language is not impossible; however, the structure of the language and its related processor are restrictive enough that it is more efficient to start over. Therefore, the requirements of a generalized test system should be considered, which will include the basic necessities previously described. This will ensure that the test language will accommodate the test actions necessary to conduct the test program regardless of the test article (a Saturn or a Shuttle).

Table 1 Comparison of Typical Operators

ACTION	ATOLL	ATLAS	OTHER	RECOMMENDED
1. Apply or turn on: a. an analog stimulus b. a discrete stimulus c. a digital stimulus	N/E* SSEL, DISO, MDSO N/E	APPLY APPLY APPLY	STIMULATE (7, 8)** TURN-ON (4, 5) APPLY (7, 8) LINK (8)	APPLY, SET, TURN, or SEND
2. Acquire the value of: a. an analog parameter b. a discrete parameter c. a digital parameter	DELY, TEST, READ DELY, TEST, SCAN DELY, TEST	MEASURE MEASURE MEASURE	CHECK/ANALOG (7, 8) CHECK/DISCRETE (7,8) LINK (8)	MEASURE, READ
3. Open the circuit connecting the Unit-Under-Test (UUT) and the test system	N/E	OPEN	TURN ON (5) SET (7, 8)	SET----OPEN
4. Close the circuit connecting the UUT and the test system	N/E	CLOSE	TURN OFF (5) RESET (7, 8)	SET----CLOSED
5. Select connection for routing signals between test system equipment UUT test points	N/E	CONNECT	CONNECT (6, 7, 8)	[Connection included in APPLY and MEASURE statements]
6. Remove connection for routing signals between test system and UUT test points	N/E	DISCONNECT	DISCONNECT (6) RESET (7, 8)	[Removal of connection included in APPLY and MEASURE statement]
7. Vary signal input until measurement satisfies required condition	N/E	ADJUST	N/E	[Macro capability will satisfy requirement when needed]
8. Determine acceptability of acquired values	N/E	COMPARE	IF (4, 5)	IF
9. Acquire and compare	SCAN	VERIFY	CHECK/ANALOG (7, 8) CHECK/DISCRETE (7, 8) CHECK/PCM (7, 8) IF (4, 5)	VERIFY
10. Repetitively acquire and evaluate - - display if no-go and branch (single values or multiparameters)	MNTR DELY	MONITOR and DISPLAY (only) VERIFY	DO and DI MNTR (4) MONITOR (4, 5)	EVERY (time units) VERIFY (function)
11. Acquire the value of several samples of a parameter and store	N/E	N/E	SAMPLE (4, 5)	N/E
12. Perform arithmetic operations	N/E	CALCULATE	Arithmetic Assignment Statements (3, 4, 5) CHECK/ANALOG (7, 8) ADD, SUBT, MULT, & DIV	LET (variable reference equal numeric formula)
13. Display tutorial, informational, or error messages	DPLY DPYM DFLG RECD DMON RGMT RCDC	DISPLAY INDICATE	DISPLAY (4, 5) DISPLAY (7, 8) PRESENT (5)	DISPLAY (variable) INDICATE (fixed)
14. Display descriptions and associated slides to operator	DMON	N/E	DISPLAY MA (6) DISPLAY (4)	DISPLAY [Canned Message]
15. Record output on line printer or typewriter	RECD RDY RLP	PRINT	DEVICE - PRINT (5) PRINT (4)	PRINT
16. Record output on magnetic tape, drum, or disc	RECD RDY RMT	RECORD	DEVICE - TAPE (5) RECORD (4)	RECORD
17. Save data for later high speed retrieval	READ RGMT RCDC SETT	SAVE	READ (5) SAMPLE (4) SAVE (7)	READ
18. Invoke or call a test program	EXEC CALL EXEM	N/E	START (3) BEGIN (4, 5) SEQUENCE (7, 8) EXECUTE (4)	PERFORM PROGRAM
19. Conditional transfer	INCX TFLG MTFG TEST SCAN DELY	GO TO----IF	MEASURE (7, 8) CK/ANALOG (7, 8) CK/DISC (7, 8) IF----THEN (4, 5)	IF----THEN

Notes: *N/E - No equivalent

**1. ATOLL 4. ATOLL-X 7. CTL 10. ASEP
2. ATLAS 5. MOLTOL 8. VTL 11. STOL
3. CLASP 6. TOOL 9. ADAP

Table 1 (concl)

ACTION	ATOLL	ATLAS	OTHER	RECOMMENDED
20. Unconditional transfer	GO TO	GO TO	GO TO (3, 4, 5) RETURN (4, 5)	GO TO
21. Transfer control to the operator	SEMI SEMI-R	WAIT FOR (operator intervention)	HOLD, STOP, HALT (4, 5) INTERROGATE (4) REQUEST (5) (part of operating system) (7)	REQUEST
22. Repeat step or group of steps imbedded in program	EXEC SEMI (operator choice)	REPEAT	REPEAT (7, 8)	REPEAT
23. Provisions for concurrent testing	N/E	N/E	START (4, 5, 6)	CONCURRENTLY PERFORM
24. Provisions for synchronizing two separately conducted test programs	N/E	N/E	SYNC (4, 5)	SYNCHRONIZE
25. Exit from present program temporarily to provide for other languages	EXEM CALL	LEAVE and RESUME	ENTER ASSEMBLY CODE (4, 5) DIRECT and END (3)	LEAVE and RESUME
26. Identify a routine to be executed as a result of an interrupt	TERM	N/E	POST (4, 5) ON (3) INTERRUPT (10) POST SIM (11)	WHEN INTERRUPT (interrupt name) OCCURS PERFORM (program name)
27. Enable/disable interrupts	N/E	N/E	POST SIM (11)	ENABLE DISABLE
28. Postpone execution until time event or value occurs	DELY 1. time 2. event 3. value	DELY WAIT FOR	DEFER/KEY (7) DEFER/TIME (7) DELAY (4, 5, 8, 10) WAIT (4, 5)	WHEN (time) AFTER (time) VERIFY (event or value) ----WITHIN (time)
29. Return system to quiescent state prior to additional testing	N/E	FINISH	N/E	[Black box approach of airlines makes this operator attractive for their application]
30. Change program statement	N/E	ALTER	N/E	[Undersirable from an operational viewpoint]
31. Establish a series of statements to be accomplished within a specific time period	N/E	PREPARE & EXECUTE	PROC and EXECUTE (4, 5) IMMED UNTIL (4)	WHEN (time critical subroutine) (See para 3.18)
32. Communication between two or more digital machines	N/E	N/E	REQUEST and TRANSMIT (4) DISPATCH (5) LINK (8) DIRECT (3)	SEND and READ
33. Subroutine delimiters	BEGIN and RETN	DEFINE and END	PROC and EXIT (3) BEGIN and END (4) PROC and END (5)	BEGIN and END
34. Program delimiters	NAME and END	BEGIN and TERMINATE	START and TERMINATE (3)	BEGIN PROGRAM and PROGRAM COMPLETE
35. Provide standard values for one or more characteristics of a signal type	N/E	SPECIFY	N/E	N/E
36. Assign a name to a specific function or signal	DECL	DEFINE	DECLARE (4, 5)	SPECIFY REPLACE (substitute an abbreviation)
37. Declare lists, tables, or names, for stored parameters	RGMT RCDC PROB PROC	N/E	DECLARE ARRAYS, LISTS, & STRINGS (4, 5)	DECLARE
38. Include a block of common statements or routines into the program as desired	MLSR	N/E	INCORP (4, 5)	[Macro capability provides this capability]
39. Predetermined lists of discretes which will be legal during program run	DISA	N/E	DOMASK (4)	[Identify in dictionary data bank]
40. Specify which display consoles will be enabled to effect program operation	CODE	N/E	CONSOLE (4) LEGAL (10)	[Test operation and program can be structured to ignore inadvertent console action]
41. Remove or add specific or all discretes from a monitor profile	PREM PROC	N/E	DOMASK (4) DIMASK (4) RELEASE MONITOR (4, 5)	ACTIVATE DEACTIVATE RELEASE

3.1 Test Action Operators

3.1.1 Apply or Turn On (Table Item 1) - The application of a stimulus includes the application of a voltage (ac or dc) of a proper amplitude (and frequency if ac). Some systems require only 28 vdc, which limits the variety of necessary stimuli. Others require ac (sine wave, square wave, ramp, sawtooth, etc) with various frequencies, amplitudes, and accuracies. It is necessary at this point in the development of a language, to ensure that a capability for complex stimulus requirements can be met, and that we do not settle for the application of discretes only. The selection of a term to accomplish this function crosses engineering disciplines. The test engineer may: "apply" a certain voltage to a test point; "turn-on" ac power; open or close a valve; set or reset a flip flop; or activate or deactivate a function. It is desirable to provide terms that are familiar to each discipline. Manual test procedures were analyzed in an attempt to select such test action operators. The recommended terms for applying a stimulus are APPLY, SET, SEND, or TURN-ON, i.e., APPLY an analog voltage, SET a discrete, SEND a digital command, TURN "AC POWER" ON. The most appropriate of the allowed terms for the application of a signal to the system under test is left to the test engineer's choice. In the case of the Shuttle, stimulus generators will be built into the interface units (IU) associated with each line replaceable unit (LRU). Analog, discrete, and digital stimulus requirements will have to be addressed during the execution of tests.

3.1.2 Acquire the Value (Table Item 2) - The acquisition of a test value includes several operations depending upon the operator. Measure, by itself, implies acquiring the value but does not indicate what should be done with the measured value. The least that can be done is to store the value in a register or in memory for later consideration. READ a clock, a register, or an input value implies the same meaning.

3.1.3 Switching Operations (Table Items 3, 4, 5, and 6) - The application of stimulus and the acquisition of a measurement normally require switching matrices for the routing of signals to and from the UUT. Operators selected to accomplish these functions are SET- -OPEN and SET- -CLOSED. In addition, set-up connections are accommodated in the SPECIFY statements. The test systems engineer will be responsible for providing this information. Such switching capability is provided to ensure that the language will satisfy the program regardless of the UUT or the test system.

For purposes of the Shuttle, all communications between the UUT and the test system will use digital transmission. All conversation of digital-to-analog signals will take place in the UUT input/output units (I/O). Measurements of the UUT systems will be digitized and transmitted back to the test system over digital transmission busses. Application, acquisition, setup, comparison, and the decisions to be made are all subject to the nature of the UUT, the test system, and the test philosophy. The test language structure should provide the flexibility necessary to accommodate a Titan, a Saturn, or a Shuttle test system. The alternative is a language that provides a capability for a unique program and becomes outmoded upon completion of the program.

3.1.4 Vary Signal Input (Table Item 7) - There are occasions, during calibration or alignment, that necessitate adjusting the signal input to a test article until the proper output is achieved. ATLAS provides this capability with the operator ADJUST. Rather than add an additional term to accomplish this action, it was felt that the provisions for macro programming would accommodate this feature.

3.1.5 Acceptability of Acquired Values (Table Item 8) - The ability to compare the acquired value with predetermined limits is required by all test systems. The selection of "IF _ _ _ _ THEN" to provide this feature was chosen rather than the COMPARE of ATLAS as it included the decision provision in an explicit manner rather than implicitly as it occurs in ATLAS. The action operator VERIFY also provides a compare feature.

3.1.6 Acquire and Compare (Table Item 9, 10) - "Verify" implies acquiring a measurement, comparing the value to determine if satisfactory, and making a decision about what should be done if the value is not acceptable. MEASURE, COMPARE, and DECISION are included in the term "Verify," which is a multiaction operator. There are times when many parameters are to be verified. They are of interest to the operator only if found to be out of tolerance or out of limits. The time of interest may be of concern. For example, in the Stage I-C of the Saturn, there are many critical parameters continuously monitored. As long as the values of these parameters are within limits, no display, printout, or operator attention is required. However, during the course of prelaunch checkout, a test period exists where the first stage separation has been simulated. The monitor for the first stage is no longer of interest and a new monitor is exercised. For this reason, time, measure, compare, and decision are included

in a monitor capability. The number of times that parameters should be interrogated becomes of interest. The statement "EVERY (time period) VERIFY - - - - -" has been selected to implement the monitor capability. Single parameters or multi-parameter tables may be monitored with this operator.

3.1.7 Single Parameters/Multiple Samples (Table Item 11) - No specific term has been provided to accommodate this feature. ATOLL II and MOLTOL provide SAMPLE to accomplish this action. ALOFT handles this operation with a macro or subroutine if such a requirement develops. It is believed that multiple samples of multiple parameters is a more realistic requirement and is provided by the EVERY (time value) VERIFY _ _ _ _ _ operator. This operator includes comparison and decision as opposed to SAMPLE, which acquires and saves only.

3.1.8 Arithmetic Operations (Table Item 12) - An arithmetic capability has been included in the most recently developed test languages (ATOLL II, MOLTOL, and ATLAS). Conversations with NASA test engineers have indicated the desirability of an arithmetic capability. Such a capability would have reduced the number of machine language subroutines presently used in the SATURN test program. The alternative to providing this capability is to encourage the use of machine coded test sequences. This will result in the sacrifice of the readability and self documenting capability provided by the near-English test oriented language.

3.2 Display or Store Operators

The next group of operators to be discussed emphasizes what to do with acquired information. These choices include display, print, record, or store. It is conceivable that a test system which uses this test language may not have a line printer, a tape recorder, or a microfilm projector. If a test system lacks hardware functions that compromise the full capabilities of the language, a subset, excluding the unusable terms, would be used. It is better to provide the flexibility during the development of the language than it is to attempt to modify it after the syntax has been completed and the language processor has been developed.

3.2.1 Output Operators (Table Items 13, 14, 15, and 16) - Two terms have been selected for display: "INDICATE" and "DISPLAY." INDICATE refers primarily to light, while DISPLAY is reserved primarily for variable displays (CRT, PLASMA, ALPHANUMERIC, E/L, etc). Many operators included in the test languages analyzed

caused information to be presented. For example, "RCDC" in ATOLL could cause the countdown clock time to be presented to the operator if the test writer had included a "1" in the condition field. It was obvious to the test writer after a short acquaintance with the language, but it was never apparent to the reader. An attempt has been made to make all characteristics of the language explicit. If a display is desired, DISPLAY or INDICATE are the only operators that will present information to the test operator. If a printer output is desired, "PRINT" must be requested. "RECORD" refers specifically to the recording of data on a magnetic device (tape, drum, or disc).

3.2.2 Store or Save Operator (Table Item 17) - In the normal course of test execution, data is acquired and stored temporarily. This action is provided by the "READ" operator. Other languages use such terms as READ, SAVE, and SAMPLE in addition to those that involve an implicit store. READ/SAVE or READ/STORE involve multiaction operators. The selection of one operator to provide multiaction is done to reduce and to simplify the number of action operators that must be remembered by the test writer and the test reader. The choice appears to be arbitrary, but a quantity must be read if it is to be saved and for this reason our choice was READ.

3.3 Program Control Operators

3.3.1 Program Invocation (Table Item 18) - Program control operators provide the test environment with the flexibility necessary to provide alternate paths during the execution of a test. Operators such as START, BEGIN, CALL, EXECUTE, or PERFORM are used to call or invoke a test program and are common to most test languages. The selection of one word over another appears to be arbitrary. Our choice is PERFORM and it was selected because it offers the least confusion with normal program delimiters.

3.3.2 Transfer of Control (Table Items 19, 20, and 21) - Transfer of control during the sequential operation of a test is a useful and necessary operator. The decision capability is provided by this operator. In ATOLL, the conditional transfer was implied in at least six operators. ATOLL-II, MOLTOL, and ATLAS had specific conditional transfer operators. Most all of the languages provided an unconditional transfer operator: GO TO. Where several paths may be taken as a result of some unique condition, an operator is needed that eventually sends all branches back to the main path. For these reasons, a definite need is demonstrated for both a conditional and an unconditional transfer.

Another form of transfer of control exists where operator intervention is necessary. In some instances, the operator must provide action or a choice as a result of a program anomaly. For these reasons, some means are required for operator intervention. The term REQUEST has been selected for this action.

3.3.3 Test Execution Operators (Table Items 22, 23, and 24) - Three operations related to test execution are REPEAT, CONCURRENTLY PERFORM, and SYNCHRONIZE.

ATOLL provided a "repeat" capability through operator choice following a SEMI. In ATLAS, CTL, and VTL this capability is provided by an explicit operator. Whether explicit or implicit, the need to repeat a test or a step is required for numerous reasons, e.g., trend data can establish the rate at which a condition is changing; if a transient has affected a measurement, repeating it often gives the desired result; a test may be repeated in order to determine an average.

The number of items requiring verification, and the short time to be provided for prelaunch test operations indicate that concurrent testing will be a necessity for the Shuttle program. Techniques will be required that will minimize the time required. The earlier test languages did not provide for conducting tests concurrently. Test languages such as ATOLL-II, TOOL, and MOLTOL include this capability. START was used to execute tests concurrently. As many as four separate tests could be conducted concurrently with TOOL. Our choice of operator for this action is felt to be more explicit: CONCURRENTLY PERFORM.

The ability to synchronize concurrent tests was provided only by ATOLL-II and MOLTOL. This capability enables multiprograms to get to the same point in the test sequence by placing an operator such as SYNC (ATOLL-II and MOLTOL) or SYNCHRONIZE (ALOFT) in the sequence. A specific program is held at that point until the other concurrent programs reach that location. Further testing will be dependent on getting all programs to the SYNCHRONIZE statement before continuing.

3.3.4 Interruption or Postponement of Test Sequence (Table Items 25, 26, 27, and 28) - Most test languages provide for leaving the test language and allowing another language to be used. Machine language programs are used predominantly to provide capabilities not required originally by the test language. The most English-like of the operators was provided by ATLAS with LEAVE and RESUME.

These operators have been selected for ALOFT. The inclusion of machine language capabilities within the ALOFT program will be accommodated with these operators.

Most test languages have included a capability for performing a safing routine as a result of a situation or condition that may become hazardous. A program is initiated to reduce or eliminate the possibility of danger to the test article or the individuals in proximity. The interrupt was transparent in ATOLL when the program labeled TERM was initiated. Posted routines were used by ATOLL II and MOLTOL. INTERRUPT was the choice of ASEP. A multiaction operator, which is explicit, has been selected for ALOFT: WHEN INTERRUPT (interrupt function name) OCCURS PERFORM (subroutine or program name). Many interrupt conditions can exist during a prelaunch test. Some are of concern during the same time period, while concern for others is scattered throughout the test. After certain functions have been completed, their related interrupts are of no consequence. A few remain of importance throughout the test period. For these reasons, two additional operators are required: ENABLE/DISABLE INTERRUPT (interrupt name). For the period of time when a specific interrupt is of importance, the capability is provided for enabling the interrupt. As soon as the period of concern has passed, the interrupt may be disabled.

In the majority of test situations, a requirement exists for postponing the execution of a test. It may be necessary to allow switching transients to settle prior to making a measurement. Another requirement normally exists for postponing the test program until some event occurs or until a parameter reaches a specified value. All test languages provide this necessary capability. ALOFT provides these capabilities with WHEN GMT IS (time value) THEN PERFORM _ _ _ or AFTER CDC IS (time value) THEN APPLY _ _ _. The use of WHEN and AFTER is specifically related to time. To provide the capability for postponing action until an event occurs or until a test value is reached, the VERIFY (statement) WITHIN (time value) OTHERWISE _ _ _ statement is used. The terms selected are more English-like than DELY & DEFER/KEY, and provide more flexibility than WAIT FOR.

3.3.5 Finish and Alter (Table Items 29 and 30) - ATLAS provided the operator FINISH, which provided an opportunity to return the test system to the quiescent state before initiating a succeeding test program. The airline test situation is geared primarily toward testing black boxes, rather than complete systems. This operator allows the removal of a test article before connecting

another test article and rerunning the test program. While useful for airline practices, it was not felt to be of value to this language; therefore, it was not included.

ATLAS is the only language that provides for changing or altering a program statement after the translation process has been completed. ASEP provided for changing parameter values in the control room after the compiling process was complete. Test engineers in general feel that these operators are useful. Management level individuals feel that this capability is neither necessary nor desirable. Modifications, additions, or deletions should be submitted through normal channels and proper approval loops considered. Providing the test operator with the means to change statements or to vary parameter values was not included in ALOFT for these reasons. During test program development this feature is valuable, but not during prelaunch operations.

3.3.6 Execution with Time Limitation (Table Item 31) - With ATLAS, a series of statements to be accomplished within a specific time can be established. The series of statements is first defined as a procedure. The action operator then EXECUTES (procedure) WITHIN (time). The PROC and EXECUTE features of MOLTOL and ATOLL II do not provide the time limitation. ATOLL II provides an IMMED/UNTIL operator that commands the system to execute the statements immediately following the IMMED/UNTIL. This means "as rapidly as possible" UNTIL such time as a Boolean expression goes true, an END of a block is reached, or until after execution of the "step label," which appears following the "UNTIL." This feature provides a capability for conducting time critical sequences that have been a part of most missile and space test programs. ALOFT is capable of executing a routine or a statement at a critical time. If the subroutine includes time limiations on the period of operation, then it is a time critical subroutine. This can be handled with the "time prefix" feature without adding additional action operators. ALOFT also provides for identifying a subroutine as CRITICAL, in which case it will be executed "immediately" without interruption.

3.3.7 Communication between Digital Machines (Table Item 32) - A digital communication feature is normally required whenever the test article includes a computer and the test system is computer controlled. Most test languages developed to date rely on machine language code to perform this function. Using this technique causes the program to lose all readability, and consumes time to code the program. ATOLL-II, MOLTOL, and VTL recognized the need for digital communication between computers, and they provide

program control operators to accommodate this feature. ATOLL II was primarily concerned with transmission and reception of digital data between the two ground computers, which are components of the Saturn test complex. MOLTOL was primarily concerned with communication between three airborne computers. VTL was concerned with loading an airborne computer from a ground computer and verifying the loading operation. The number of processors presently planned for the Shuttle indicates a need for digital communication between the ground and the airborne computers. Test writers have indicated a desire to write their airborne computer test programs in an English-like language. This feature would improve the total readability of the ALOFT test procedure over that provided with a machine language program. SEND and READ operators are provided for the application of a stimulus and the acquisition of the resulting responses. Digital information identified by function name can be transferred to and from the vehicle with these operators. Therefore, no new terms or syntactic units were required to accommodate this feature.

3.4 Language Processor Directives (Table Items 33 and 34)

Two types of block structures are accommodated in ALOFT: PROGRAMS and SUBROUTINES. BEGIN PROGRAM and PROGRAM COMPLETE and BEGIN and END provide the boundaries that mark the beginnings and the ends of programs and subprograms respectively. The languages studied make use of program, procedure, do, or begin and end delimiters. Program boundaries (delimiters) serve to inform the compiler of start and finish points for processing, and provide entry and exit points that can be identified by the test operator. BEGIN and END were selected for subprogram delimiters because they are more English-like than PROC and END, DEFINE and END, or BEGN and RETN. The selection of the program delimiters BEGIN PROGRAM and PROGRAM COMPLETE, rather than those terms used by other test languages (NAME and END, BEGIN and TERMINATE, or SEQUENCE and END SEQUENCE) were also made because they are more English-like.

3.5 Declarations (Table Items 35, 36, and 37)

ATLAS provided a capability for defining standard values for one or more characteristics of a signal with a SPECIFY statement. For example: voltage type, range, incremental provisions, tolerances, and ac component may be specified in the SPECIFY statement. Thereafter, in the program statements, reference need be made only to the voltage type appearing in the SPECIFY statement.

```
XXXXX  SPECIFY, SOURCE, DC SIGNAL, VOLTAGE 28V ERRLMT
       + - 4V, A C - COMP .001V $
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
YYXXX  APPLY, DC SIGNAL, CNX HI J2-3 LO J2-4 $
```

This type of information must be included in the Function Specification in ALOFT. The SPECIFY statement in ALOFT provides the final tie between the language, the test system, and the test article. As has been previously emphasized, ALOFT is independent of the test system and the test article. The SPECIFY feature of ALOFT affords the capability of tying the test system and the test article together through the English-like language. Through the use of SPECIFY, functions, connections, addresses peculiar to either test system or test article, and any subroutine required to accomplish the conversion can be identified. The SPECIFY statement also provides the test writer with function names that are in the vernacular of the disciplines with which he is associated. In the case of Shuttle, LRU and IU designers will select the function names to be used later by the test writer. To further aid the test writer (at no sacrifice to the reader) the operator "REPLACE" is provided, which permits the substitution of abbreviations for previously defined, more involved character strings. The test writer must indicate in the program coding that he is using an abbreviation for a previously defined term. The language processor senses the abbreviation and substitutes the original character string. The full spelling is restored to ensure that readability has not been impaired. New names may be substituted for previously used names with this operator.

DECLARE (names, lists, and table) has been included in ALOFT. The ability to declare names and assemble data into lists or tables provides a technique that is difficult to equal without considerable loss in operating time, increased coding effort, and reduced efficiency. Establishing a table for regularly scheduled monitoring permits selecting when to monitor, at what intervals, when to release, and making modifications to the table during the course of the test.

3.6 Macro Provisions (Table Item 38)

ATOLL II and MOLTOL provided an operator that enabled a number of common statements, used at many intervals throughout the program, to be written once and then incorporated into the program at the necessary intervals. This operator was INCORP. A unique operator such as INCORP was included in ALOFT as it was felt that this capability will be valuable after the test article requirements have been identified. ALOFT makes use of a MACRO to accomplish this feature.

3.7 Provisions for Controlling Inadvertent Action (Table Items 39 and 40)

ATOLL and ATOLL II ensure that inadvertent discrete operation will not affect the running of a test. DISA or DOMASK identify the discretes to be acknowledged during the program. If the test writer calls for a discrete that has not been established previously by DISA or DOMASK, the compiler will detect it and indicate an error. The writer will be required to add that discrete to his preamble. This feature is provided by ALOFT through the use of the dictionary data bank. All stimuli issued to the vehicle must be specified in the dictionary data bank. The routing instructions and addresses are included in the specification.

Another feature included in ATOLL, ATOLL II, and ASEP is that of specifying the consoles that will be allowed to affect program operation, or will be provided with displays concerning the program. Terms such as CODE, CONSOLE, and LEGAL identify specific consoles associated with the test. This feature is provided by the capabilities inherent in the dictionary data bank. All output devices with their routing instructions and addresses must be specified prior to program initiation. Each CRT, line number, printer, and magnetic recording device must be specified.

3.8 Monitor-Profile Construction and Modification

The removal or addition of discretes from a monitor profile can be accommodated by specifying a table and ACTIVATING or DEACTIVATING the previously defined functions in the table. The DOMASK provided this feature in ATOLL II while PREM and PROC accomplished the task in ATOLL. Previously, all items to be considered (analog or discrete) during the test must have been declared in the Table Declaration.

4.0 LANGUAGE SPECIFICATION SUMMARY

4.1 General

This section summarizes the major features of ALOFT (detailed in the Appendix). Also, applications of these features are discussed.

The Appendix uses syntax diagrams to illustrate the construction of all legal elements of the language from the basic characters and symbols to complete programs. This technique was chosen because it is precise, minimizes the ambiguities associated with prose descriptions, is more condensed than prose descriptions, and facilitates rapid comprehension of alternative statement constructions. (The technique is an adaptation of the ATLAS specification ARINC SPECIFICATION 416-1.)

An attempt has been made to minimize the number of rules and limitations applicable to the use of syntactical definitions; however, there are areas where some logic is necessary to correctly use the diagrams. For example, a diagram might allow the assignment of text constants where numeric values have been called for. Expansion of the diagrams to eliminate all possibility of erroneous usages would adversely affect rapid comprehension of the more essential features of the element represented by the diagram. The language processor would be expected to detect most erroneous assignments of this type.

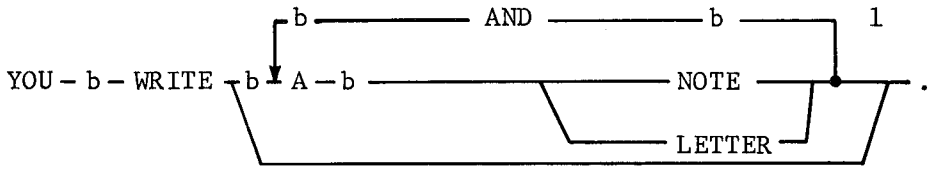
The specification is not intended to be a training manual for the language, but would be expected to be used frequently as a reference manual by both program writers and readers. The syntax diagrams facilitate this application. The explicitness of statements constructed in accordance with the diagrams should minimize the necessity for readers to use reference materials to understand the resulting programs.

One of the major features of the language is the provision to allow "names" to be assigned to functions, tables, lists, variable quantities, subroutines, programs, etc. The selection of these names has a significant impact on the readability and understandability of the resulting programs. It is therefore anticipated that management and control of many of the selections would be exercised by the project. There should be correlation between names appearing in interface specifications, schematics, block diagrams, signal lists, etc, and the names appearing in function name specifications in this language. The use of functionally

descriptive names should generally be encouraged for the sake of the reader. As will be discussed later, the writer may optionally abbreviate long names to reduce writing time but the longer names will appear in printouts.

4.2 Syntax Diagram

Syntax diagrams are similar to flow diagrams in that they show the legal sequence of items, including any alternative branches and iterations. As an example, the syntax diagram



would allow any of the following sentences to be written:

- YOU WRITE;
- YOU WRITE A NOTE;
- YOU WRITE A LETTER;
- YOU WRITE A NOTE AND A LETTER;
- YOU WRITE A LETTER AND A NOTE.

The applicable diagram usage rules are as follows:

- ——— is simply a connecting path;
- b is a blank space;
- capital letters and characters are to be used as shown;
- diagonal lines represent alternate forward paths;
- vertical lines represent connectors for a return path or loop that is optional;
- a number on the beginning vertical line of a loop indicates the maximum number of times the loop may be used.

The Appendix refines these rules and defines all legal characters and symbols used in ALOFT syntax diagrams.

4.3 Basic ALOFT Statements and Statement Prefixes

As with English and most higher level programming languages, the lowest meaningful and complete element of ALOFT is a statement. Within ALOFT statements, words and phrases are inserted to help readability and prevent misinterpretations and errors by users. These are generally verbs, articles, prepositions, etc, which make the statements English-like. They are required to be used precisely as shown in the syntax diagrams. The complete statement, rather than a single word, defines the action or purpose of the statement. In general, however, a *verb* or *operation code* in the statement is a very strong indication of the type of activity or purpose of the statement. In addition to basic statements, ALOFT has provisions for including optional *prefix phrases*, which may be either a condition for execution of the statement or an action to be performed at essentially the same time.

The basic statement and prefix phrase types, as indicated by their key words, are listed below. The parenthetical notes are included to further explain the associated actions.

		APPENDIX PARAGRAPH
Send actions		
APPLY	(Analog or digital function)	2.5.2
SET	(Discrete functions, valves, clocks)	2.5.2 2.5.1
TURN	(Discrete functions on or off)	2.5.2
SEND	(Digital data)	2.5.2
Acquire actions		
READ and SAVE	(Discretes, clocks, digital)	2.5.3 2.5.1
MEASURE and SAVE	(Analog, digital)	2.5.3
VERIFY	(Read or measure with conditional transfer)	2.5.7
Invocations or calling statements		
PERFORM	(Subroutine	2.3.9
PERFORM PROGRAM	(Program)	2.3.9
EXECUTE	(For macros only)	2.3.9
USE	(Data bank)	2.4

		APPENDIX PARAGRAPH
Delimiters		
BEGIN	(Program, data bank, subroutine)	2.3.10 2.4 2.3.8
MACRO	(Beginning of macro definition)	2.4
COMPLETE	(Program, data bank)	2.3.10 2.4
LEAVE ALOFT	(To another language)	2.4
RESUME ALOFT	(After leaving ALOFT)	2.4
Interrupt manipulation		
WHEN INTERRUPT	(to identify action as a result of a named interrupt).	2.5.5
ENABLE	(Interrupt)	2.5.5
DISABLE	(Interrupt)	2.5.5
Sequence control		
GO TO	(Unconditional transfer)	2.3.7
IF	(Variable reference conditional transfer)	2.3.7
VERIFY	(Function conditional transfer)	2.5.7
WHEN INTERRUPT	(See above)	2.5.5
REPEAT	(Single statement)	2.3.7
Assignment or arithmetic operation		
LET	(Variable reference) = (Value or formula)	2.3.6
ASSIGN	(Variable reference) = (Discrete or Boolean state)	2.3.6
Concurrent program implementation		
CONCURRENTLY PERFORM	(For concurrent programs)	2.5.6
SYNCHRONIZE (n)	(Synchronization points in each program)	2.5.6

		APPENDIX PARAGRAPH
Prefix phrases and timing control		
WHEN (Clock=time)	(Precedes action statement)	2.5.1
SET (Clock=time), AND	(Precedes action statement)	2.5.1
AFTER (Clock=time),	(Precedes action statement)	2.5.1
STATEMENT (number)	(Statement label where required)	2.3.7
Other time phrases		
---WITHIN (Time value)	(To set a time limit for VERIFY)	2.5.7
---FOR (Time value)	(To generate a timed discrete or pulse)	2.5.2
Operator interfaces and records		
DISPLAY	(Messages)	2.5.4
INDICATE	(Lights or fixed states)	2.5.4
PRINT	(Variable messages)	2.5.4
RECORD	(Variable messages)	2.5.4
REQUEST	(DISPLAY message then READ and SAVE keyboard input)	2.5.4
Definition statements		
SPECIFY	(Function)	2.4
DECLARE	(Table, list, internal variable)	2.3.4
BEGIN	(Subroutine, program, data bank)	2.3.8 2.3.10 2.4
REPLACE	(Abbreviation, Substitution)	2.3.8
MACRO	(Macros)	2.4
Miscellaneous		
ACTIVATE --	(Acknowledge or honor a function in a table)	2.3.6
DEACTIVATE --	(Ignore a function in a table)	2.3.6

Some isolated statements, as they might appear in a printout of a program or subroutine, are listed below. Definition type statements are discussed later.

SET _CDC TO -1 HR, AND APPLY _RUDDER 2 CONTROL POSITION_ +14.5DEG. WHEN _CDC IS -58MIN, MEASURE _RUDDER 2 ACTUATOR HYDRA TEMP_ AND SAVE AS _RA2 HYD TEMP_.

IF _RA2 HYD TEMP_ IS LESS THAN 400DEGF THEN GO TO STATEMENT 20.

STATEMENT 10 INDICATE _HYDRAULIC TEMP WARNING_.

REQUEST TEXT (WHAT NOW? (STOP) (GO ON)) ON _CRT 1, LINE 1_ AND SAVE INPUT AS _OP IN_.

IF _OP IN_ IS EQUAL TO TEXT (STOP) THEN PERFORM _RUDDER 2 POWER SHUTDOWN_.

STATEMENT 20 GO TO STATEMENT 33.

STATEMENT 33 EVERY 30SEC, VERIFY _RUDDER 3 ACTUATOR HYDRA TEMP_ IS LESS THAN 350DEGF OTHERWISE GO TO STATEMENT 10.

TURN _VOR RCVR 2 POWER CMD_ OFF.

VERIFY _VOR RCVR 2 POWER MNTR_ IS OFF WITHIN 5SEC OTHERWISE GO TO STATEMENT 135.

PERFORM PROGRAM _ELEVATOR 2 CHECKOUT PROGRAM_.

STATEMENT 3095 EVERY 3MIN, VERIFY _IMU 2 ROTATION DET_ FUNCTIONS ARE BETWEEN _UPPER LIMIT_ AND _LOWER LIMIT_ OTHERWISE GO TO STATEMENT 4051.

AFTER CDC IS -5MIN, RELEASE STATEMENT 3095.

Statement 3095 above is actually a monitor of all of the functions listed in the table called _IMU 2 ROTATION DET_. Each function in turn is measured and the measured values are compared with values in the table columns called _UPPER LIMIT_ and _LOWER LIMIT_. Inclusion of the word TABLE in the name of the table would have greatly speeded the recognition of the activity.

The class of statements referred to previously as *definition statements* are of several types. In general, they do not connote actions to be performed by the system but are used to define or declare to the language processor and (usually) readers what a subsequently used *name* will mean or will stand for. Each type of definition statement is discussed below.

The SPECIFY statement is used to specify the location, type, (address codes) and the applicable conversion subroutine for a system function or signal, and to formally assign a *name* to the function so described. Since the SPECIFY statements are peculiar to test system implementation and require detailed knowledge of the test system and its programming, they will be used only in dictionary data banks prepared by test system engineers.

There are three types of DECLARE statements. The first is a simple data declaration, wherein a *name* (and memory location) is reserved for a specifically defined type of data. The data itself may optionally be *assigned* a value by the declaration statement. In all subsequent usages, the *name* refers to the data contained in the reserved location, never to the location itself. The second type of DECLARE defines and *names* a LIST of data with an *index* (i.e., subscript). The index may be used to identify individual data values in the list. The third type of DECLARE *names* a TABLE having rows and *named* columns of data with specifically defined characteristics. One column of every table is called FUNCTION and contains *function names*. An unlimited (but declared) number of other columns contain units and (optionally) such values as upper limit, state, last value, and time. Tables and their uses are discussed in paragraph 4.5.

The BEGIN and MACRO statements are the initial statements of program structures of multistatement groups including *programs, subroutines, macros,* and *dictionary data banks*. They assign names and characteristics to the structures and are discussed in later paragraphs.

The REPLACE statement is used to identify substitutions that the program writer wishes the language processor to make before the program is processed. There are two useful ways that this substitution capability can be used. The test writer may wish to abbreviate or number (with a few characters) all of the *names* to be used in his program, then write his program using the abbreviations. He might also desire to abbreviate frequently used language primitives such as the word STATEMENT. To do so he would first instruct the language processor with the following statement:

```
REPLACE 'S' WITH (STATEMENT).
```

Subsequently, his program would be written:

```
'S' 3124 APPLY - - - -.
```

After processing, the above would read:

STATEMENT 3124 APPLY - - - -.

Another application of the REPLACE statement is to *change* a *name* used in a program. With this statement, the language processor can be instructed to:

REPLACE _NAME 1_ WITH _NAME 2_.

The desire to do this may be the result of incorrect usage of _NAME 1_ when the program was originally written, a change in the name of a function, or perhaps an original program could be used for another application if one or more names were changed wherever they appeared in the program.

4.4 Program Structures

ALOFT program structures or multistatement groupings are of several types, to meet various requirements. They are called programs, subroutines, macros, and dictionary data banks.

4.4.1 Program - A PROGRAM is the highest structural grouping and is completely executable in its own right, providing only that substructures used by it have been defined. One PROGRAM may call for the execution of other programs, with the simple restriction that a called (lower level) program may not call for the execution of any of its calling (higher level) programs. No specific restriction on the number of levels is contained in the language rules. The calling statement is of the form:

PERFORM PROGRAM *program name* .

A provision is made in the language to call for the concurrent execution of the called program while the execution of the higher level program continues. The calling statement would then be as follows:

CONCURRENTLY PERFORM PROGRAM *program name* .

In the event that synchronization points are required in the concurrently executed programs, each of the programs must contain identical

SYNCHRONIZE n.

statements at the desired synchronization points. (n is the number of the synchronization point, and unless an identically numbered synchronization point occurs in some concurrent program,

the statement will be ignored). The first program(s) to reach a common synchronization point will wait until all programs reach it before continuing. The initial statement of a PROGRAM is of the form:

BEGIN PROGRAM_UNIQUE PROGRAM NAME_.

The final statement is:

UNIQUE PROGRAM NAME COMPLETE.

For each program, an initial group of statements, sometimes referred to as a preamble, will identify the data base required for the execution of the program. This data base includes specifications and declarations of several types. Some such data will be extensive and previously defined for use in any number of programs. These groups of data, called dictionary data banks, are discussed later, and are identified by an assigned name.

Other data may be uniquely declared (defined) by the program writer for use in the program. In the latter case, the various types of declaration statements are used.

It should be noted that *function specifications* can be accessed *only* through dictionary data banks.

4.4.2 Subroutine - A subroutine is a group of statements to perform some activity, test, sequence, or routine that can be uniquely defined as an entity and may be repeatedly required in one or more programs, but cannot be performed unless called for by a program (or another subroutine of a program). It will have a unique name to identify it. ALOFT allows values and variable references to be passed to the subroutine, and retrieved from it, by the calling program.

One major application of the subroutine will be to define a procedure to be executed in the event of a hazardous equipment failure. The subroutine might perform safing functions, then display, print and record the status of the system. This type of subroutine might be assigned high operational priority in the system. ALOFT allows a subroutine to be identified as CRITICAL, in which case it will be performed without interruption.

Subroutines will use data bases provided by their calling programs. New data acquired or generated by the subroutine will be carried back to the calling program only when the calling statement PERFORM identifies the outputs.

Examples of subroutine calling statements are as follows:

PERFORM_COORD TRANSFORM ROUTINE_ WITH INPUTS _A_, _B_, _C_, 45DEG, 10DEG, AND 0DEG, AND OUTPUTS _X_, _Y_, AND _Z_.

WHEN INTERRUPT _ OVERTEMP_ OCCURS PERFORM _FIRE HAZARD WARNING_.

VERIFY _ FLAPS POSITION_ IS GREATER THAN 50PCT OTHERWISE PERFORM _ FLAP EXTENSION ROUTINE_ WITH INPUT 55PCT.

The initial statement of a subroutine is both a delimiter and a declaration of the subroutine name. The inputs and outputs identified in the declaration and in the PERFORM statements must be in identical order and number.

BEGIN_UNIQUE SUBROUTINE NAME_ WITH INPUT_SUBSYSTEM NAME_ AND OUTPUT_SUBSYSTEM STATUS_.

BEGIN CRITICAL _SHUTDOWN SEQUENCE_ WITH INPUT _DISPLAY AREA ASSIGNED_.

The corresponding final statements are:

END _UNIQUE SUBROUTINE NAME_.

END CRITICAL _SHUTDOWN SEQUENCE_.

4.4.3 Macro - A macro is similar to a noncritical subroutine from a test writer's viewpoint, but it will never appear on a final program printout. (It might appear on interim working copies if requested.) Instead, the sequence of statements represented by the macro will be substituted by the language processor just as if the writer had so prepared the program.

The name of a macro will be a character string, as with other names, but it will not have underscores as delimiters.

Macros may be defined or declared by the writer for use in his program, or they may be a part of a dictionary data bank used (initially) by the program.

As with a subroutine, input values and both input and output references may be assigned by the macro invocation statement. However, since the macro boundaries are discarded by the language processor, all internal variables of the macro must be a part of the calling program or subroutine. A macro invocation or calling statement might be:

```
EXECUTE ADJUST _RUDDER 2 VDA_, 0.5MA, 5.8MA, 1.0MA/SEC,
0.1MA, _RUDDER 2 POSITION_, 5.0DEG, _R2 VDA MA_, _ADJ
FLAG_.
```

The initial macro statement, both a delimiter and a name declaration for the macro, might be:

```
MACRO ADJUST _ADJUSTED FUNCTION_, _INIT VALUE_, _MAX VALUE_,
_ADJ RATE_, _INCREMENT_, _MEAS FUNCT_, _LIMIT VALUE_,
_FINAL VALUE_, _STATUS_.
```

The final statement would be:

```
END.
```

In general, the "dummy" names of the macro declaration statement and within the macro action statements would be indicative of the information to be inserted by the program writer in the calling statement. These dummy names would be replaced by the values and functions from the calling statement when the program is processed as well as when the printouts are made. As stated previously, the above three statements would be used only by the test writer and would not appear in final source code documents. The prime purpose of the macro is to enable potentially useful routines to be named and documented for reuse or incorporation in other programs with a minimum amount of formality. The test writer is totally responsible for determining whether the current macro "fits" and is usable in the program. Once the program gets through an initial pass of the language processor, the source program is totally independent of the macro and its possible changes. Programs and subroutines, on the other hand, must have rigid configuration control because they will still appear in source program documentation, and they may be separately coded and stored in the operating system.

4.4.4 Dictionary Data Banks - A dictionary data bank is a grouping of specifications and/or declarations that have been assigned a group name. These data banks can then be referenced or used by programs that need them. The contents of a specific dictionary

data bank might include all function specifications (SPECIFY statements), standard variable reference declarations (DECLARE statements), standard table declarations (DECLARE TABLE statements), and standard list declarations (DECLARE LIST statements) that are associated with the checkout of a particular subsystem or a particular phase of a mission or test. In addition, the dictionary data bank might include appropriate subroutines and other programs, and one dictionary data bank might require the use of other dictionary data banks.

In general, the using project would manage and maintain a library of dictionary data banks that could be used by programs as needed.

ALOFT rules will allow a program to call for an unlimited number of dictionary data banks. Project management, however, may wish to restrict the scope of a program by allowing only specific dictionary data banks to be used. Since function specifications can appear *only* in dictionary data banks, the "allowable" functions for actions by the program can be controlled. For configuration control purposes the use of groups of subroutines and programs can be tracked and managed through the dictionary data bank concept.

It would most probably be desirable that reference data banks of MACROS and REPLACE statements be separately identified and made available. They would require very little configuration control due to their usage. Printouts of all dictionary and reference data banks would generally be available to all program users.

4.5 Tables

Special attention has been given to the definition and use of TABLES. They should prove to be a significant aid in test program preparation and visibility. The TABLE declaration statement is relatively long, and may at first seem quite complex. The flexibility and usefulness of table operations, however, warrants this concession. It is expected that most table declarations will be made by more experienced programming personnel, and that such declarations will be included in dictionary data banks. The test program statements that deal with tables are relatively explicit once the table structure is understood.

4.5.1 Table Definitions - An ALOFT TABLE may be represented by Figure 1. The items in bold type are fixed for all tables, items in italics are definition terms, and items in regular type are to be assigned by the table declaration statement.

The column values may subsequently be reassigned by action and assignment statements.

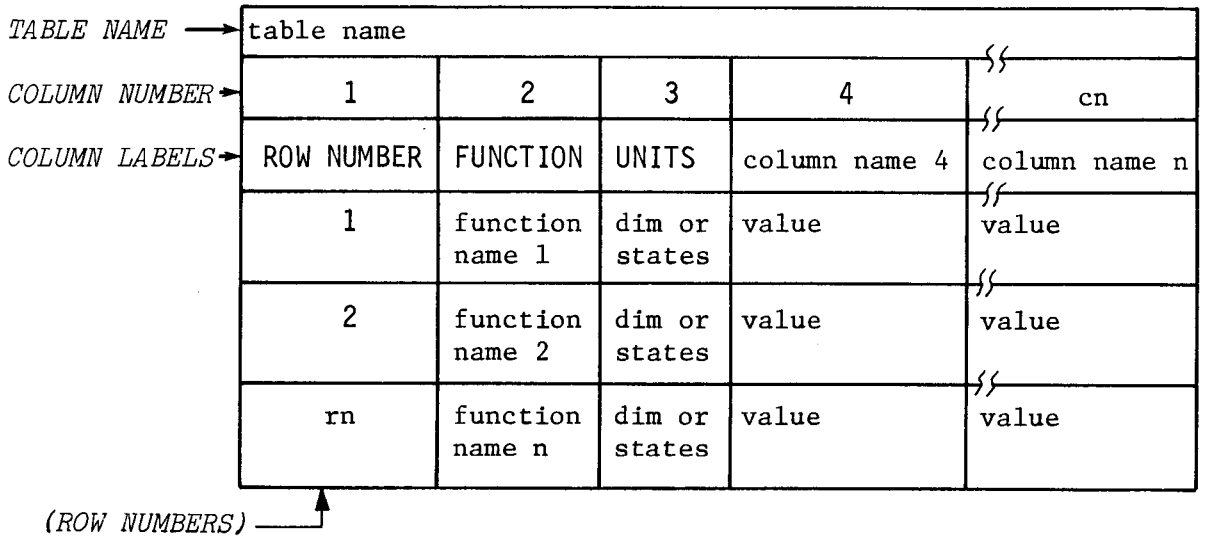


Figure 1 ALOFT Table Format

The *function names* assigned to the FUNCTION column must be specified in dictionary data banks. The function specifications will include the identification of the type of signal through the named conversion routine. The UNITS column of the table will identify the dimensional units (or allowable states) of the function values (or states) to be placed in later columns of the table.

The column numbers and row numbers of the table are fixed when the number of rows and the number of columns are declared. Subsequent references to the table contents may be made through the use of row and column numbers, function name and column name, or index numbers. The row and column index numbers are themselves assigned names that can be dealt with as variable references in "step-through" and looping operations.

Figure 2 presents a DISCRETE table.

DISCRETE MONITOR PROFILE 12				
1	2	3	4	5
ROW NUMBER	FUNCTION	UNITS	_REF_	_LATEST_
1	_PROP STATUS LOX CHILLDOWN_	ON/OFF	OFF	ON
2	_LOX PREVALVE_	OPEN/CLOSED	CLOSED	OPEN
3	_FUEL TANK OVERFILLED_	TRUE/FALSE	FALSE	TRUE
4	_ENGINE 4 HEATER_	ON/OFF	OFF	ON

Figure 2 Discrete Table

The table illustrated above would be declared by the following statement:

```
STATEMENT 32691 DECLARE TABLE_DISCRETE MØNITØR PRØFILE 12_WITH 5 CØLUMNS INDEXED
                BY_CN_AND LABELED RØW NUMBER, FUNCTIØN, UNITS,_REF_BØØLEAN,
                _LATEST_BØØLEAN, HAVING 4 RØWS INDEXED BY_RN_WITH ENTRIES
'' ,            2                    3          4          5                  PI
  _RN_      FUNCTIØN                 UNITS      _REF_      _LATEST_           AND
  1,  _PRØP STATUS LØX CHILLDØWN_,   ØN/ØFF,    ØFF,       ØN                 AND
  2,  _LØX PREVALVE_,                ØPEN/CLØSED, CLØSED,  ØPEN               AND
  3,  _FUEL TANK ØVERFILLED_,        TRUE/FALSE, FALSE,    TRUE               AND
  4,  _ENGINE 4 HEATER_,             ØN/ØFF,    ØFF,       ØN                 ;
```

The optional blank spaces and comments inserted in this declaration have contributed to the readability of the table entries. It is most probable that the writer would first have made a table much as shown in Figure 2 before writing the statement. The format of the declaration statement printout would be similar in appearance, and would facilitate rapid checking.

Figure 3 represents a table of analog functions. Its declaration would naturally include more columns and rows, but otherwise would be similar to that for the discrete table.

RUDDER CONTROL TEST 3 TABLE					
1	2	3	4	5	6
ROW NUMBER	FUNCTION	UNITS	_UPPER LIMIT_	_LOWER LIMIT_	_LAST VALUE_
1	_RUDDER 1 HYD FLUID TEMP_	DEGC	315	-50	0000
2	_RUDDER 2 HYD FLUID TEMP_	DEGC	315	-50	0000
3	_RUDDER 1 VDA TEST INPUT_	MA	+13.5	+13.0	0000
4	_RUDDER 2 VDA TEST INPUT_	MA	+13.5	+13.0	0000
5	_RUDDER 1 ACTUATOR POS_	DEG	+ 5.3	+ 5.0	0000
6	_RUDDER 2 ACTUATOR POS_	DEG	+ 5.3	+ 5.0	0000
7	_RUDDER 1 POS REF VOLTAGE_	V	+10.1	+ 9.9	0000
8	_RUDDER 2 POS REF VOLTAGE_	V	+10.1	+ 9.9	0000

Figure 3 Analog Table

The table column names in the illustrations are perhaps typical of the majority of the tables that would be used. Additional columns might be added to save values of the functions at various times during a test or mission, and to save time tags associated with the readings.

The previously described tables are associated with the acquisition and evaluation of data. Tables may also be desirable to establish sets of values and/or discrete states of functions to be applied at essentially the same time. The table would be constructed in identically the same manner, except that column names would change and the functions named would be stimuli and commands. Such a table could be particularly useful to establish a "safe" condition, to reset a subsystem status after each of a series of tests, or to establish a condition connoted by "power up all systems."

4.5.2 Use of Tables - Except for the table declaration statement, the only other statements that deal exclusively with tables are ACTIVATE and DEACTIVATE, which are used to mask and unmask table

entries on a row basis. While a table row (function) is deactivated, it is "frozen" and ignored by all other statements referring to the table. The following statements might appear anywhere in a test program:

```
DEACTIVATE_DISCRETE MONITOR PROFILE 12_ _LOX PREVALVE_.

ACTIVATE_DISCRETE MONITOR PROFILE 12_ ROW (4).

ACTIVATE_RUDDER CONTROL TEST 3 TABLE_ _RUDDER 1 POS REF
VOLTAGE_.
```

Many ALOFT statements can use tables. This is accommodated by allowing either *function name* or *table name* FUNCTIONS to be used in send action and acquire action statements, or by allowing variable references to include table entries.

As an example, an APPLY statement can be

APPLY *table name* FUNCTIONS *column name.*

which will be interpreted as a command to apply each activated function named in the table with the value (or state) shown in its entry under *column name.* Of course, SET, SEND, or TURN could be used instead of APPLY. Also, a desired value or state could be given rather than referred to the column entries. The statement

TURN *table name* FUNCTIONS ON.

would turn all of the active table functions on.

An acquire action statement would be as follows:

MEASURE *table name* FUNCTIONS AND

SAVE AS *column name.*

This type of statement would measure all active functions in the table and save their values (or states) in the indicated column. If a time column had also been declared for the table, the following statement could be written:

READ GMT INTO *table name column name.*

This statement would place the current GMT value in all active row entries under the time column name.

An individual entry can be referred to by *table name function name column name*, thus allowing the statements:

> READ CDC INTO *table name function name column name.*
>
> LET *table name function name column name* EQUAL -3HR 33MIN 10SEC.
>
> ASSIGN *table name function name column name* OPEN.
>
> LET_IMU TEST LIMITS TABLE_ _RATE GYRO 3 DRIFT_ _UPPER LIMIT_EQUAL _CURRENT VALUE_ * 1.3.

In the last statement, note that the table and row (or function) identification has not been repeated for _CURRENT VALUE_, since they are assumed to be the same as for _UPPER LIMIT_. In general, the rule is that the language processor will make this conclusion if there is a column by that name in the table previously named in the statement. If not, another variable reference (not in a table) will be assumed.

The VERIFY statement, which includes both acquisition and evaluation of functions, can be used to implement scans and monitors of table functions. These statements would include the following forms:

> VERIFY *table name* FUNCTIONS ARE *column name* OTHERWISE GO TO STATEMENT *number.*
>
> VERIFY *table name* FUNCTIONS ARE GREATER THAN *column name* OTHERWISE PERFORM *subroutine name.*
>
> STATEMENT 416 EVERY *time value* VERIFY *table name* FUNCTIONS ARE BETWEEN *column name 1* AND *column name 2* OTHERWISE PERFORM *subroutine name.*
>
> VERIFY *table name* FUNCTIONS ARE ON WITHIN *time value* OTHERWISE PERFORM *subroutine name.*
>
> IF *variable reference* IS ON THEN VERIFY *table name 1* FUNCTIONS ARE *column name* OTHERWISE GO TO STATEMENT 312.
>
> WHEN CDC IS *time value*, VERIFY *function name* IS OFF OTHERWISE VERIFY *table name* FUNCTIONS ARE *column name* OTHERWISE DISPLAY ----.

The IF statement will call for the execution of a VERIFY if an internal variable such as a flag is on. The last statement would execute the table verification only if *function name* (which might be an operator's switch) is ON. The DISPLAY statement

would be executed only if the subsequent table verification failed. Note that regardless of the actions accomplished within this statement, the next following statement will eventually be executed.

5.0 WRITING, DOCUMENTATION AND CHECKING AIDS

ALOFT has been designed for ease of use by the (subsystem) design engineer/test writer and for nonambiguous understanding by the various readers of the language. To accomplish this, the language uses English-like statements specifically fitted to the education, vocabulary, experience, and training of the personnel from the many technical disciplines that will use the language.

5.1 Writing Aids

The programs prepared in ALOFT can be written by the engineer involved in the design of the system, subsystem, or LRU, or by a test writer designated to prepare the test program. To help in this task the following writing aids are provided.

5.1.1 Natural English Statement Structure - The language syntax provides a natural English-like sentence structure for writing the tests. The language is designed with a minimum number of rules and restrictions. The specifications in syntactic diagram format provide an easy-to-use guide to the writing of test procedures in ALOFT.

5.1.2 Dictionary Data Banks - The test writer will rely on dictionary data banks to provide the specific names of functions for the equipment he is to test or operate. These function names, for the most part, will be the same as those appearing on the drawings, data lists, etc, for the equipment under test. A system design engineer will normally identify the functions by name and define their characteristics to a test system designer, who will then prepare appropriate SPECIFY statements for the functions and include them in the dictionary data bank.

5.1.3 No Coding of Terms - The printouts and source language records will not have coded terms and abbreviations except as contained in names. The users will not have to remember specific codes to write or understand tests.

5.1.4 Abbreviations - To shorten the writing task of the test writer he may define any abbreviations he chooses. The compiled listing of the test will show the complete English words or statements he had elected to abbreviate. For example, to keep from writing PROPULSION CONTROL SYSTEM 1 many times in his test program the writer could use the abbreviation 'PCS 1'. When he had finished writing the program he would instruct the compiler to

REPLACE 'PCS 1' WITH (PROPULSION CONTROL SYSTEM 1).

Then everywhere he had used 'PCS 1', PROPULSION CONTROL SYSTEM 1 would be printed out.

5.1.5 Name Substitutions - A capability of substituting one name for another is available to the test writer. This capability can be used to advantage to prepare similar test programs for redundant devices.

When the test writer knows that he will have to repeat the test for several devices, he will use a dummy name such as "PROP MTR TEST' in writing the text. Then before compiling, a control card such as

REPLACE 'PROP MTR TEST' WITH (PROPULSION MONITOR TEST 1)

would be added to the preamble. This would cause PROPULSION MONITOR TEST 1 to be substituted throughout the test for PROP MTR TEST.

The test could then be compiled again with a new control card;

REPLACE 'PROP MTR TEST' WITH (PROPULSION MONITOR TEST 2)

in the preamble. This would cause the test to be compiled with PROPULSION MONITOR TEST 2 substituted throughout the test for PROP MTR TEST.

Another capability of name substitution is the ability to change a name in the data dictionary and not have to rewrite tests. For example, assume ALPHA in the data dictionary is changed to BETA. To use a program previously written using ALPHA, the writer would place a control card in the preamble stating

REPLACE_ALPHA_WITH_BETA_.

BETA would be substituted for ALPHA throughout the program.

5.1.6 Subroutine Capability - To keep from writing repetitive routines the writer defines the series of statements as a subroutine. To call the subroutine he writes

PERFORM SUBROUTINE *NAME* WITH - - - (any inputs or outputs required).

5.1.7 Tables - Writing tests in table form can help minimize the writing effort. For example, the writer can generate a table with limits or states for the various functions in the table. He can then write a test to confirm that all functions in the table are within limits or match the profile by writing

VERIFY *TABLE NAME* FUNCTIONS - - -

5.2 Documentation

The ALOFT language, in conjunction with a properly designed compiler, will provide self-documenting capability.

5.2.1 Readability - The use of English-like statements ensures nonambiguous readability of the language.

5.2.2 Abbreviations - The writer can establish any abbreviations he desires to ease the task of program preparation. The compiler will produce full listings with proper substitutions for all abbreviated portions of statements.

5.2.3 Comments - It is possible for the test writer to intersperse comments throughout the test program to ensure that the reader understands what is taking place. This capability will not generally be needed but is available for the use of the test writer.

5.2.4 Data Dictionary - The heart of the ALOFT language is its data dictionaries wherein technology-oriented names are placed and defined. The dictionary concept permits any unit under test function to be defined in terms of the test system.

5.3 Checking

Checking will be facilitated by the naturalness of statement structures and the self-documenting capability of the language.

5.3.1 Syntax Designed for Ease of Checking - The syntax has been designed with checking requirements in mind. With the test procedure readily understood, all users can readily verify that the program does what is desired.

5.3.2 Error Detection - This is primarily a compiler function, but a properly designed compiler used with ALOFT will be able to determine syntax errors, dimensional errors, overrange stimulus, and errors that can affect performance.

The compiler will be provided with a full range of error messages to assist the test writer in preparing an acceptable program. The compiler will be designed to detect and note errors, then continue language processing. This will minimize the number of times the program must be processed before no errors are found.

6.0 TRAINING REQUIREMENTS

Training test and flight engineers to use ALOFT must be considered from two aspects. One aspect refers to the training of those who are primarily concerned with reading and reviewing existing tests; the other aspect refers to the training of those who are responsible for creating or writing the tests.

6.1 Training Test Users

The training of those who are to review the tests is minimal, since it consists of studying an introductory document containing a general description of the language. This document would describe such things as the relationship of dictionary data banks to tests and the resulting combination to the actual testing of a unit under test. No details of syntax need be available since example statements and sample tests with appropriate discussion will impart the necessary understanding to enable the reader to deal with actual tests.

Approximately two hours of study of such a document should be all the time required to fully understand the ALOFT language from the standpoint of reading and understanding existing tests.

6.2 Training Test Writers

The training of those who are to write the tests is more demanding and falls naturally into two categories. One category is the training of those who are to write the actual tests. The other category is the training of those who are responsible for the definition of the contents of the dictionary data banks.

Training of test writers should begin with the introductory document, which is the training aid for those who review tests. A training manual would subsequently provide a discussion (from the general aspects down to the syntax) of those statements and syntactical structures, with which the test writer will deal.

The syntax diagrams and semantic explanations included in the Appendix do not in themselves constitute a training manual. The syntax diagrams are relatively complex and a training manual should provide many examples of language statements to assist the trainee in understanding the meaning of the syntax.

Once he has been exposed to many examples of language statements and, as a result, has a general knowledge of the use of the language, detailed syntactic diagrams can be introduced. At this time, the trainee will be familiar with most aspects of the language and will desire the concise statement of language characteristics provided in the Appendix.

Training of those test writers who will be responsible for the definition of the contents of the dictionary data banks will begin at the completion of the training of "ordinary" test writers. This training would consist of the detailed information necessary to understand declaration statement syntax, specification statement syntax, and finally use of macro and other languages. These individuals must also possess a detailed knowledge of the operation of the test system, which is necessary because they are responsible for providing, through the dictionary data bank, the interface between the ALOFT language and a specific test system.

The time required to train a test writer should be less than a week, which could be shortened via classroom training using a knowledgable teacher. Training a test writer for definition of dictionary data banks may take a full week of classroom work. This does not include training in any specific test system. Training could be significantly reduced if the individuals involved had prior test writing or programming experience. In any case, simple programming concepts, such as flowcharting, should be introduced to all test writers to enable them to more efficiently use the language to create tests.

6.3 Alternative Approach

An alternative approach to the training of test writers is provided through the use of an off-line interactive test writing system. Cueing techniques, using on-line CRTs, are available (see Phase I report) that provide all the information needed to create test programs; no previous experience on the part of the user with respect to test writing is necessary.

A significant advantage to an interactive test writing approach is the immediate indication of errors to the test writer. As a result, corrections can be made at once, resulting in a syntactically correct program at the conclusion of a single interactive session. This represents a considerable saving in the time required to create a program, compared to older methods that require submittal of source code to a remote computer site with the attendant iterations necessary to achieve a correct program.

An additional advantage is that the approach is self teaching. No outside classwork is necessary beyond a general familiarity with the language, which is provided by the document used as a training aid for those who are to review test programs.

A disadvantage of this technique lies in the cost and time necessary to develop such an interactive system. This has to be traded off against the cost and time savings realized in test program development and the ease of training many test writers.

7.0 SUMMARY

7.1 Conclusions

ALOFT provides the higher order test-oriented language characteristics needed to test and operate the Space Shuttle and other NASA space vehicles and experiments. Using the good features of previously developed test-oriented languages and correcting for their faults; ALOFT has been designed to operate in a multidisciplined environment, independent of the test system. These important features should ensure wide acceptance by its users and permit structuring tests long before the test system is finalized.

The proposed language is readily learned, easy to write, and its English-like nonambiguous statements ensure that the readers will understand the test procedures.

The design of the language is such that it can be readily expanded or changed as conditions dictate, which further ensures long life for ALOFT.

7.2 Recommendations

It is suggested that the following studies, which are beyond the scope of this contract, be implemented:

- Verify the usability of ALOFT by rewriting existing Saturn tests (now written in ATOLL) in ALOFT. This will determine possible language shortcomings;
- Use ALOFT to write test procedures for subsystems of the Space Shuttle currently being defined. (a subsystem using many technology disciplines, such as the Space Shuttle propulsion subsystem is recommended). This study will determine the ability of ALOFT to test and operate Space Shuttle systems;
- Verify the use of "tables" and "concurrent testing" routines, as these concepts should be further proven;
- Develop a training curriculum, skill requirements, and a Training Manual for ALOFT;
- Further investigate the suggested alternative approach (interactive cueing on CRTs) to training and program preparation;
- Investigate methods of managing the dictionary data banks to determine how they can best be kept current;

- Study the management of the language and how proven test procedures can be modified to run on a modified Space Shuttle, since it is possible that each Shuttle vehicle will be different;
- Investigate the practicality of using ALOFT to verify performance of LRUs removed from the vehicle for maintenance;
- Investigate higher order languages suitable for writing the language processor;
- Write a language processor for ALOFT, after the language has been "proven" by the foregoing tasks.

The goal for the Space Shuttle to be in operation by the late 1970s and the need for a test and flight engineering oriented computer language dictates that the recommended follow-on efforts be initiated as soon as possible.

APPENDIX

SPECIFICATION OF A LANGUAGE ORIENTED TO FLIGHT ENGINEERING AND TESTING (ALOFT)

(The ALOFT Specification has been reprinted and is available as MCR-70-450.)

1.0 INTRODUCTION

A Language Oriented to Flight Engineering and Testing (ALOFT) is a high-order test-oriented computer language, designed for the checkout and operation of the NASA Space Shuttle. A degree of flexibility has been built into ALOFT, which should permit the language to be used for other space vehicles and experiments. ALOFT has been designed to be functionally independent of any test system.

Prime importance has been given to the readability of the language. Nonambiguous sentences ensure understanding of test and operating procedures written in the language, and the English-like syntactic structure of the language makes it easy to read and write, which makes it readily adaptable to all technical disciplines.

This Appendix contains the syntax diagrams and the semantic explanations (relative to the diagrams) for ALOFT. These diagrams and explanations constitute the complete specification of the language.

2.0 GENERAL CONSIDERATIONS

2.1 Format of Language Syntax Diagrams

The syntax diagrams for ALOFT are modeled after the syntax diagrams found in the Abbreviated Test Language for Avionics Systems (ATLAS), ARINC Specification 416-1, June 1, 1969.

The format of presentation used in this specification is the syntactic diagram followed by explanation of the semantics of the illustrated diagram. This combination constitutes a full definition of the structure and meaning of a language form.

An attempt has been made to clearly distinguish language characteristics from language processor or operating system characteristics. The latter characteristics are not discussed in detail, since they are properly a part of the specifications of a language processor and an operating system with which the language being defined herein would be used.

The reasoning behind the selection of those capabilities of the language, implemented as specified in this report, appears in the document *Development of a Test and Flight Engineering Oriented Language*, Phase II report, MCR-70-365.

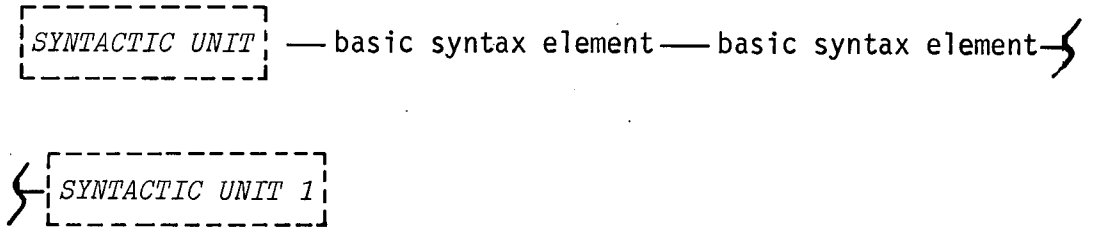
2.2 Explanation of Language Syntax Diagrams

Syntax diagrams are made up of syntactic units and basic syntax elements that ultimately reduce to the allowable letters, numerals, and symbols which make up the character set of the language. The basic syntax elements appear in syntactic diagrams as themselves or as a name that is syntactically equivalent. The syntactically equivalent name appears in lower case type. For example:

letter :: = A

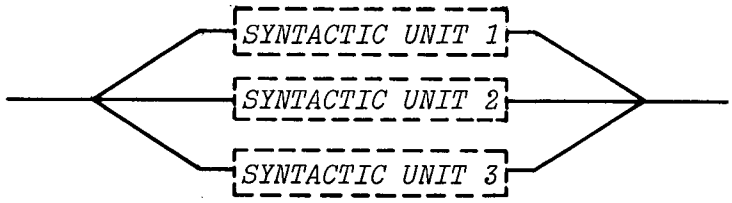
where ":: =" means syntactic equivalence. Therefore, in a syntactic diagram the construction——letter——is equivalent to the construction——A——.

A name enclosed in a dashed box is a syntactic unit defined from basic syntax elements and/or other syntactic units. A definition consists of a name within a dashed box on the left and a syntax diagram on the right. For example:



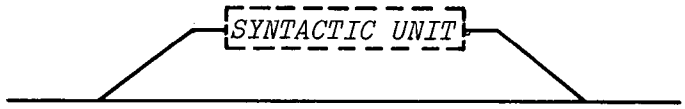
The syntax diagram in the example indicates that the syntactic unit being defined on the left is a concatenation of two basic syntax elements with a previously defined syntactic unit. The lines indicate the flow of the syntax diagram from left to right. The wavy lines indicate continuation of a syntax flow from one line on the page to the next lower line. Assume that the dashed box indicated by the name syntactic unit 1 has previously been defined as CD. Further assume that the first basic syntax element is syntactically equivalent to A and the second basic syntax element is syntactically equivalent to B. Therefore, the syntactic unit being defined on the left, is reducible to the basic syntax elements forming the character string ABCD.

Choice among syntactic units is indicated by a branching in the syntax diagram. For example:

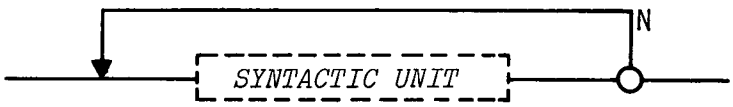


The flow of the syntax diagram illustrated allows only one branch to be taken, which results in a single syntactic unit being chosen from the three syntactic units available.

A choice between taking or omitting a syntactic unit is indicated by a branch in the syntax flow that contains no syntactic unit. For example:



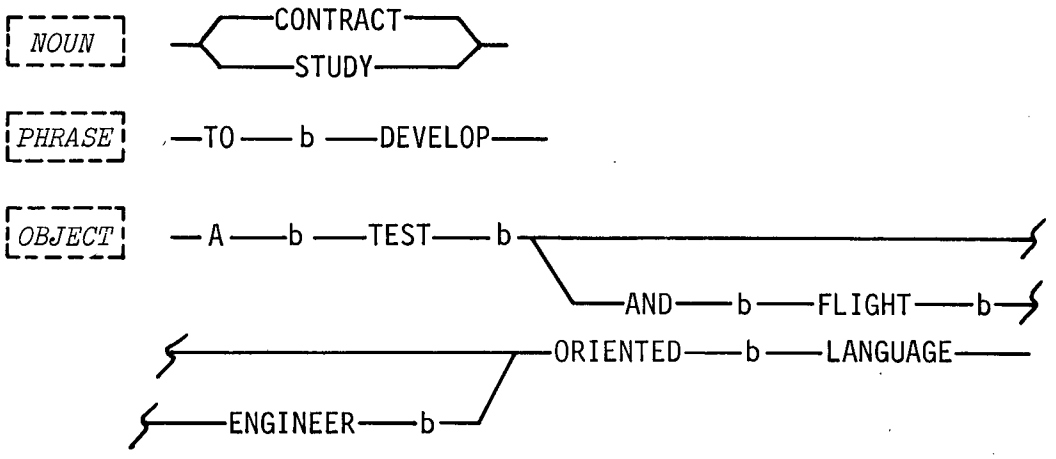
Repetition of syntactic units is indicated by a feedback loop with the maximum number of repetitions, if applicable, indicated on the loop arrow. Otherwise, the number of repetitions is undefined. A syntactic unit on a line that is part of a feedback loop must appear at least once in the corresponding statement for which the syntax diagram exists. For example:

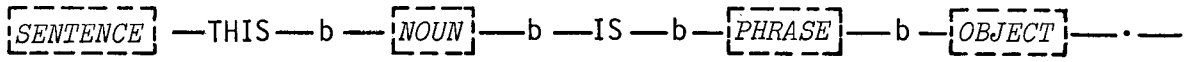


Notes that give further information on a syntax diagram appear in parentheses beneath the diagram, with an arrow indicating where in the diagram the note is to be applied. For example:

To further illustrate these concepts and to show how a syntax diagram is used to generate language statements, the following sample diagrams are presented:

b :: = BLANK





These syntactic diagrams allow the construction of the following sentences:

- THIS CONTRACT IS TO DEVELOP A TEST ORIENTED LANGUAGE.
- THIS CONTRACT IS TO DEVELOP A TEST AND FLIGHT ENGINEER ORIENTED LANGUAGE.
- THIS STUDY IS TO DEVELOP A TEST ORIENTED LANGUAGE.
- THIS STUDY IS TO DEVELOP A TEST AND FLIGHT ENGINEER ORIENTED LANGUAGE.

2.3 General Syntax Diagrams

2.3.1 Basic Syntax Elements

letter :: = A through Z

numeral :: = 0 through 9

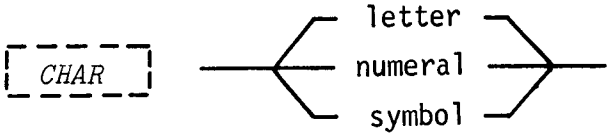
symbol :: = + - * / ( ) . , _ ' = ?
BLANK

The basic syntax element "letter" is syntactically equivalent to any one of the letters A through Z. The basic syntax element "numeral" is syntactically equivalent to any one of the numbers 0 through 9. Finally, the basic syntax element "symbol" is syntactically equivalent to any one of the symbols illustrated. Any statement or syntactic unit in the language can ultimately be reduced to the characters and symbols shown above. The method of such a reduction is detailed in the following syntax diagrams.

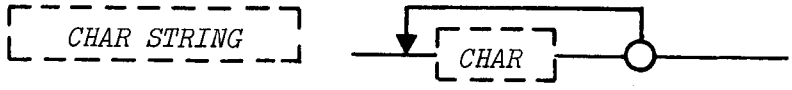
b :: = ——BLANK——

The basic syntax element "b" is syntactically equivalent to the symbol "BLANK." A "b" appearing in a syntax diagram indicates one and only one blank for each appearance of that basic syntax element.

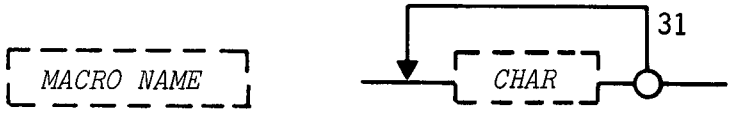
2.3.2 Simple Syntactic Units - The syntactic units defined in this subsection will be used throughout the remainder of this language specification.



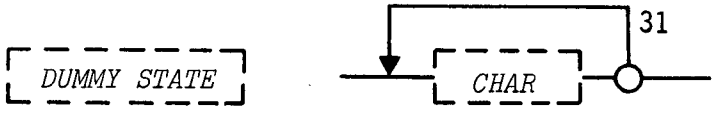
The syntactic unit "CHAR" is a choice of any single letter or number or symbol.



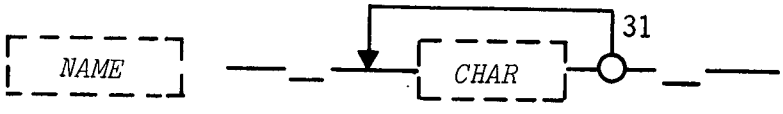
The syntactic unit "CHAR STRING" is made up of individual characters in a sequence of arbitrary length. The characters that make up the character strings represented in this manner are arbitrarily chosen by the test writer.



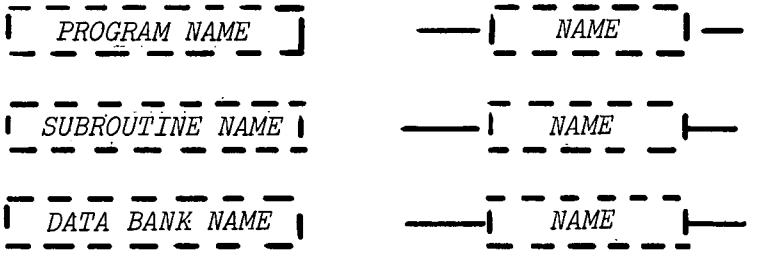
The syntactic unit "MACRO NAME" is made up of individual characters, arbitrarily chosen, in a sequence of a minimum of one character, and a maximum of 32 characters. These syntactic units name combinations of language statements that are created as extensions to the language and are referenced in a test by the name indicated by "MACRO NAME."



This syntactic unit is used in subroutine definitions (defined later) to name dummy discrete states for send action statements (also defined later).

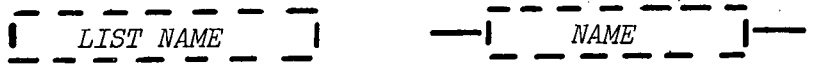


The syntactic unit "NAME" is made up of individual characters, arbitrarily chosen, in a sequence of a minimum of one character, and a maximum of 32 characters. The underscores are used to delimit the name for better reader understanding of the test in which they appear. These syntactic units name various types of data variables and program blocks (defined later), which make up both the data to be manipulated in a test and the major sections of a test.



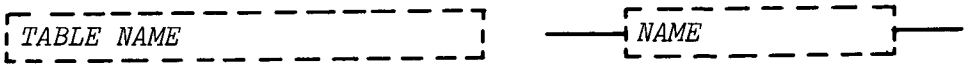
The above syntactic units name groupings of language statements (defined later) that make up major sections of a complete test.

This syntactic unit is used in the definition of a dictionary data bank (defined later) to identify a revision to a dictionary data bank which has already been defined.



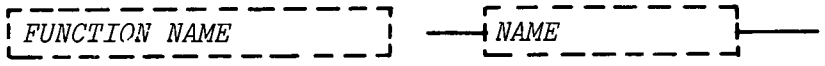
This syntactic unit is used in the list declaration (defined later) to name the list being declared.

This syntactic unit is used in the list and table declaration (defined later) to name the index attached to the list or table being declared.

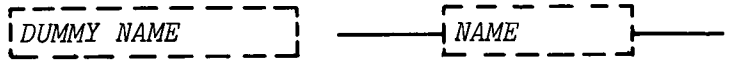


This syntactic unit is used in the table declaration (defined later) to name the table being declared.

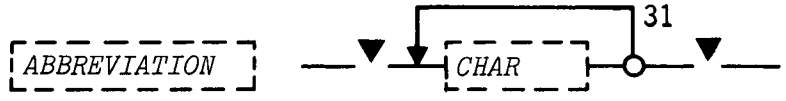
This syntactic unit is used in the table declaration (defined later) to name the column that is a part of the table being declared.



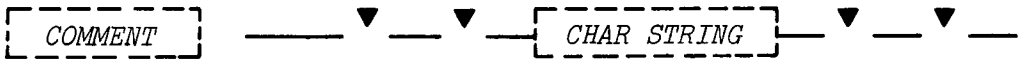
This syntactic unit is used in a function specification (defined later) to name a dictionary data bank entry for the specification of a line replaceable unit function.



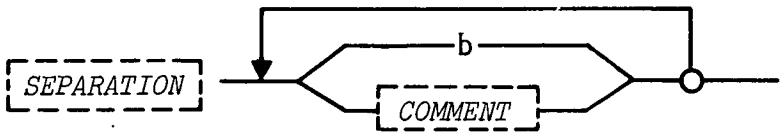
This syntactic unit is used in macro and subroutine definitions (defined later) to name dummy function names used in test action statements.



This syntactic unit is used in substitution statements (defined later) to name the abbreviation defined for use in a test as a writing aid.



The syntactic unit "COMMENT" is a character string delimited by pairs of prime symbols. No specific characters are excluded from the character string. The comment is reproduced in the language processor source listings, but is otherwise ignored in processing.



The syntactic unit "SEPARATION" is an arbitrary number of blanks and/or comments in an arbitrary sequence. At least one blank must appear where a separation is required.

The symbol "ϕ" is syntactically equivalent to the syntactic unit "SEPARATION" and will be used throughout this language specification to indicate those positions in a syntax diagram where any number of blanks and/or comments may be inserted. Such a capability permits special formatting of the source code if desired. For instance, source data may be written in a tabular format for ease of reading, without violating any language requirements.

dim :: = any of the dimensions listed in the matrix below.

FUNCTION TYPE	BASIC UNIT	$X10^0$	$X10^3$	$X10^6$	$X10^9$	$X10^{-3}$	$X10^{-6}$	$X10^{-9}$	$X10^{-12}$
volts ac/dc	volt	V				MV	UV		
current ac/dc	ampere	A				MA	UA		
frequency	hertz	HZ	KHZ	MHZ	GHZ				
	pulses per second	PPS	KPPS						
time	day	DAY							
	hour	HR							
	minute	MIN							
	second	SEC				MSEC	USEC		
resistance	ohm	OHM	KOHM	MOHM					
inductance	henry	H				MH	UH		
capacitance	farad	FD					UFD		PFD
power*	watt	W	KW			MW	UW		
	voltage, current or power	DB							
ratio	percent	PCT							
pressure	pounds per square inch	PSI							
	millimeters of mercury	MMHG							
	inches of mercury	INHG							
	millibars	MB							
distance	inch	IN							
	foot	FT							
	meter	M	KM			MM			
	nautical mile	NM							
velocity	feet per second	FT/SEC							
	meters per second	M/SEC							
	knot	KT							
	mach no.	MACH							
angle	degree	DEG							
	arcmin	ARCMIN							
	arcsec	ARCSEC							
	radian	RAD				MRAD			
	revolution	REV							
temperature	degrees centigrade	DEGC							
	degrees fahrenheit	DEGF							

*Power may also be expressed in decibels above a specified power level according to the following convention:

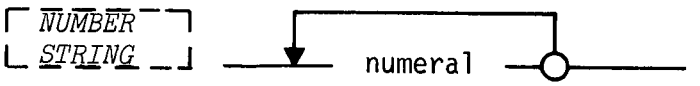
DBM is decibels above one milliwatt.
DBW is decibels above one watt.
DBK is decibels above one kilowatt.

Units of rate of change or acceleration may be formed from any of the units listed in the table. Rate units are written by adding the slash character and appropriate unit of time to the basic unit from the table. (Examples FT/SEC, V/MSEC, or REV/MIN) Accleration units are formed by adding another slash and unit of time to the rate convention. (Examples DEG/SEC/SEC, FT/SEC/SEC) The special case of acceleration, the earth's gravitational force, may be written as G in lieu of the usual acceleration convention. The following special cases of angular rate may be used:

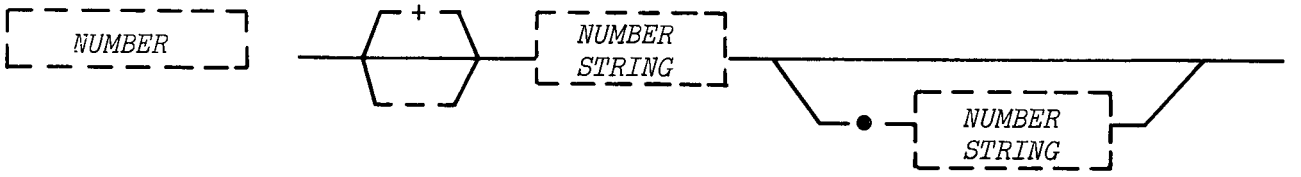
RPS - revolutions per second (in lieu of REV/SEC)
RPM - revolutions per minute (in lieu of REV/MIN)
RPH - revolutions per hour (in lieu of REV/HR)

These dimensional identifiers help to establish the internal scaling of the numeric quantities to which they are attached.

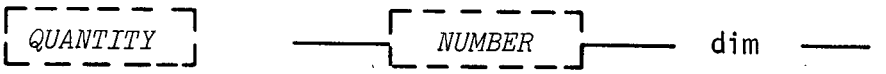
2.3.3 Data Syntax



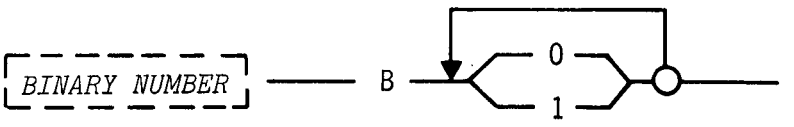
The syntactic unit "NUMBER STRING" is made up of individual numbers, arbitrarily chosen, in a sequence of at least one number and can be of arbitrary length.



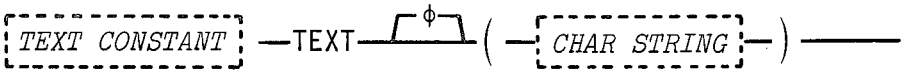
The syntactic unit "NUMBER" is carried in floating point form in a target computer, except in those areas where integer numbers are specifically required by the language.



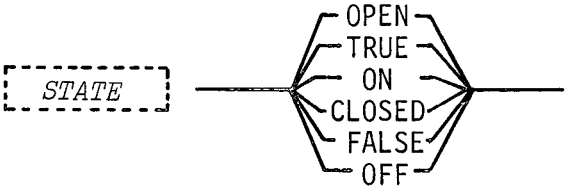
The syntactic unit "QUANTITY" is a number, with a dimension identifier attached, which provides both scaling information to the language processor and a method of error checking on the test writer's use of these quantities.



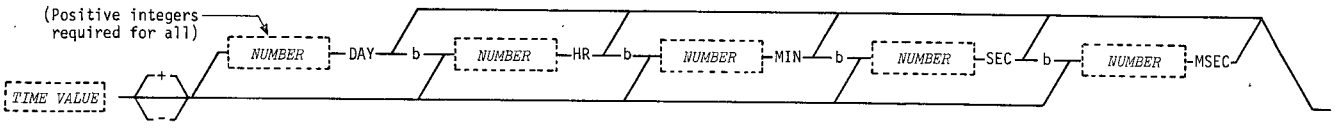
The syntactic unit "BINARY NUMBER" is made up of an arbitrary sequence of the numbers 0 and 1, of arbitrary length, and preceded by a B. The principal use of binary numbers is in the system declaration of the dictionary data bank, and as such will not generally be used by a test writer.



The syntactic unit "TEXT CONSTANT" is made up of a character string enclosed in parentheses and preceded by the word "TEXT" followed by a separation. Matched parentheses can be used within the character string with the result that no specific characters are excluded from the character string.



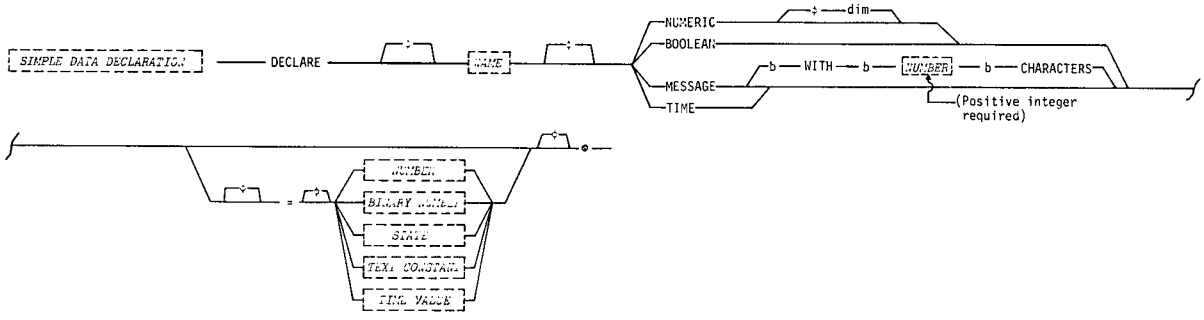
The syntactic unit "STATE" provides terms to describe the states of discretes for send action statements (defined later) and the settings of flags, which may be used in conditional transfer statements as Boolean variables (also described later).



The syntactic unit "TIME VALUE" can be written in several forms. Each form requires a consecutive series of time units (DAY, HR, MIN, SEC, MSEC) with no gaps between units. For instance, a time value which indicates DAYS followed by MIN is not legal in the language. Time value units can start at any unit and end at any unit as long as a consecutive sequence is followed. The syntactic units "NUMBER" in the time value are expected to be of positive integer form.

2.3.4 Data Variable Syntax - The variable types defined in this subsection are internal to a program and not directly related to a test article (line replaceable unit) or test system. Information of a test article or test system nature is provided in the dictionary data banks (defined later).

It is not necessary for a test writer to use these variable declarations in the creation of a test program, although he will find them useful as he becomes more familiar with the language. Variables used in a test program, but not explicitly declared in a statement of the type defined below, will be given a default syntactic structure. Such variables, which are not declared in a data dictionary bank used by the program or in a data declaration local to the program, are considered to be of "NUMERIC" type with scaling and value established by the context in which the variable is used.



The syntactic unit "SIMPLE DATA DECLARATION" is a form of language processor directive which allows a test writer to create an arbitrary name for a single variable and attach to it various data characteristics.

Numeric variables may be declared, with or without dimension identifiers. A numeric variable may be set to the data constants "NUMBER" or "BINARY NUMBER" to establish an initial value of the variable.

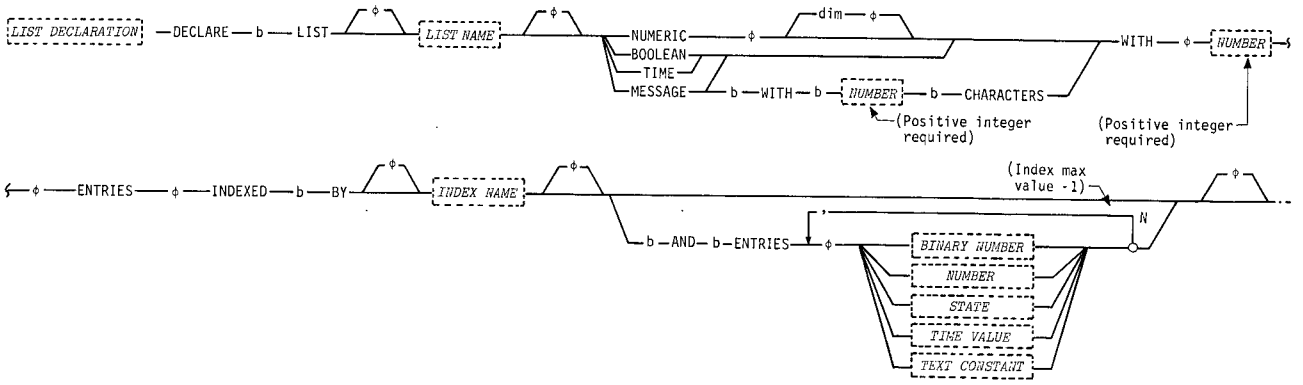
A Boolean variable may be declared for use as an internal program flag and may be set to an initial "STATE" value.

A message variable may be declared with the number of characters given as a positive integer value. The message variable may be set with a "TEXT CONSTANT," in which case the number of characters data characteristic is optional. The number of characters will be determined, in the absence of that data characteristic, by the number of characters in the text constant.

Message variables are declared for use as precanned local program messages, in which case they are initialized to a text constant value. They may also be used as reserved storage for messages to be input to a program via the read keyboard statement (defined later), in which case the number of characters expected must be declared. Equal or not equal comparisons of messages so input, with precanned messages, is allowed in the conditional transfer statement (defined later).

A time variable may be declared and may be initially set to a "TIME VALUE."

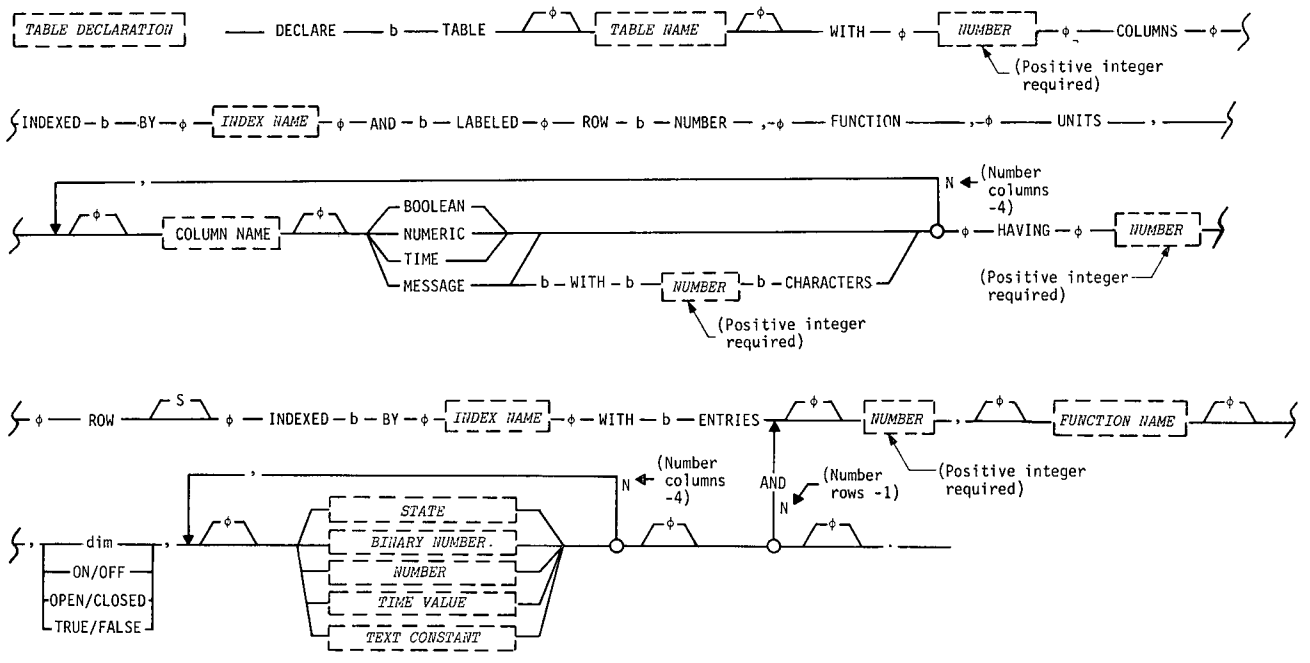
Variables initially set to an incompatible data constant value will be flagged as errors by the language processor.



The syntactic unit "LIST DECLARATION" is a form of language processor directive that allows a test writer to create an arbitrary name for a list of variables of identical data characteristics. A number, of integer form, is required to indicate the maximum number of variables in the list. The list has an index name attached to it, which is used to identify which variable in the list a test writer may want to use at a particular time.

The list of variables may have any of the data characteristics described in the simple data declaration. Boolean data variables may be used as flags to help in the control of the internal program flow.

Each variable in the list may be given an initial value compatible with the data characteristic declared.



The syntactic unit "TABLE DECLARATION" is a form of language processor directive that allows a test writer to create an arbitrary name for a table containing information pertaining to functions attached to a specific subtest. The function name, required in the second column, identifies the SPECIFY statement in the dictionary data bank which provides the information to a test system enabling the acquisition of data which is to be stored in a column of this table. Other information may reside in this table to be used for comparison purposes, limit checks, etc.

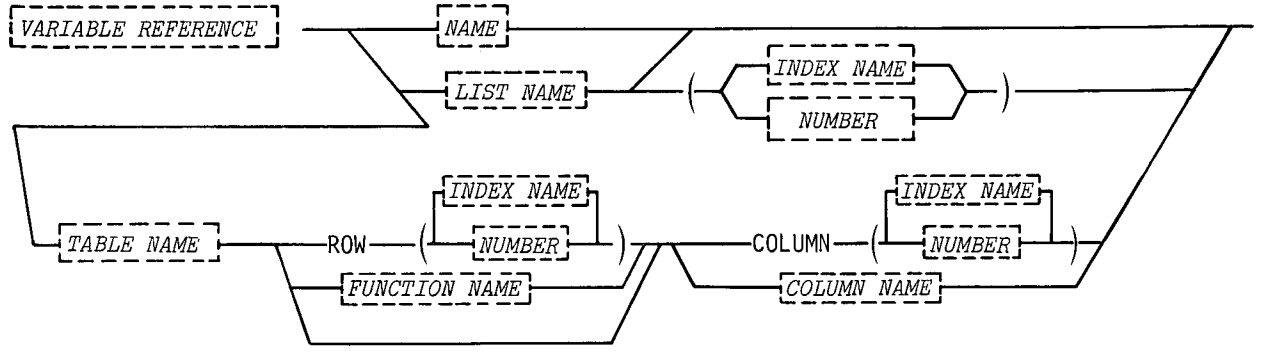
The first three columns are fixed requirements and are included to provide necessary information to both the user and language processor. These three columns cannot be dealt with by name as can the other columns of the table which have attached column names. The nature of the first three columns requires that references be made to individual rows of the column only.

The first column is an identifier for a row that is a positive integer number starting at one and increasing by one for each new row. The second column identifies a function. The third column establishes a dimension for all numeric data that appear in that particular row or the proper state terms for Boolean data. The rest of the columns may have data characteristics as described in the simple data declaration and names which enable the entire column to be dealt with as a unit.

The number of columns and number of rows must be declared as positive integers. Information entered into columns and rows must agree with the numbers declared.

An index name is defined for later use in column identification for possible explicit looping capabilities with respect to columns. Another index name is defined for later use in row identification for the same reason. Each use of an index name in a declaration statement must define a unique name.

Each variable in a row, corresponding to a column, must be initialized to a value compatible with the data characteristics declared, as described in the simple data declaration syntax.



The syntactic unit "VARIABLE REFERENCE" describes the legal ways in which variables, as declared in previously described syntactical structures, may be referenced. A single variable, declared in a simple data declaration, is referenced by the use of the name associated with the variable.

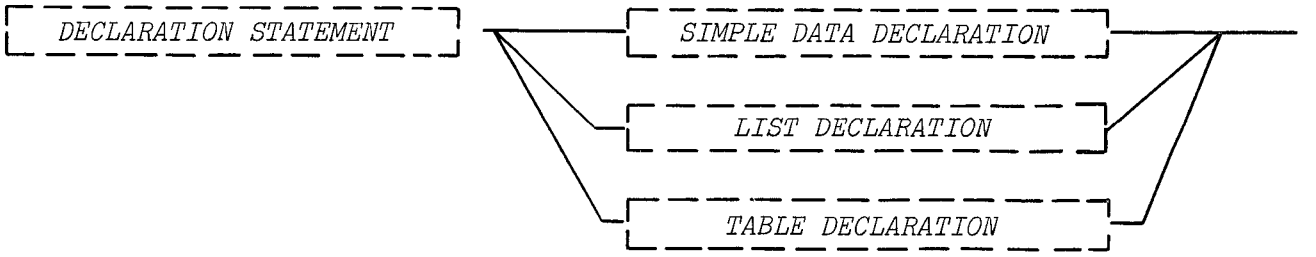
A list of variables with identical data characteristics, declared in a list declaration, is referenced by the use of the list name associated with the list. An individual member of the list is referenced by associating an index value with the list name. The index value may be a number of integer form or the index name declared in the list declaration. This index name would have had a number of integer form attached to it during the course of execution of the program.

A table, containing a number of columns and rows, each row of which contains data pertaining to a particular function, declared in a table declaration, is not capable of being referenced by itself. An individual row may be referenced by associating a row number or a function name with a table name. An individual column may be referenced by associating a column number or a column name with a table name. An individual item in a row of the table is referenced by associating a row and column identifier with the table name.

Individual items in a row are dealt with in all statements as any other simple variable would be dealt with. When columns are identified individually in a statement, then the action of the statement is applied to all individual items in that column.

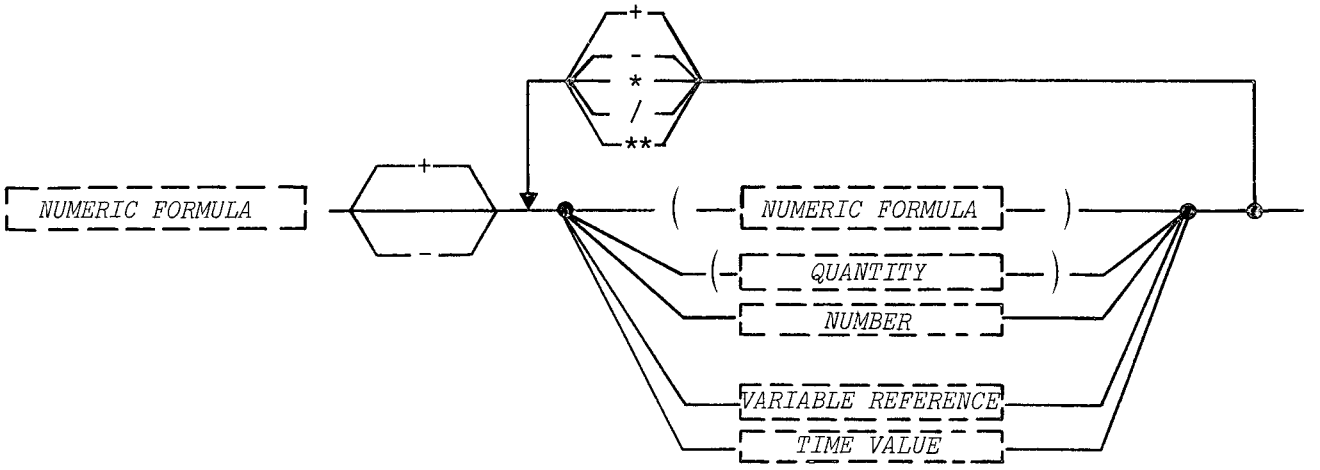
In any statement dealing with tables and their associated rows and columns, the table name need be identified only once. A column name appearing by itself in the same statement will be associated with the previously named table. If no such column name appears in the table the name will then be considered a simple data declaration identifier.

This syntactic unit is used in macro and subroutine definitions (defined later) to name dummy variable references used in the definitions.



The syntactic unit "DECLARATION STATEMENT" is a simple data declaration, a list declaration, or a table declaration.

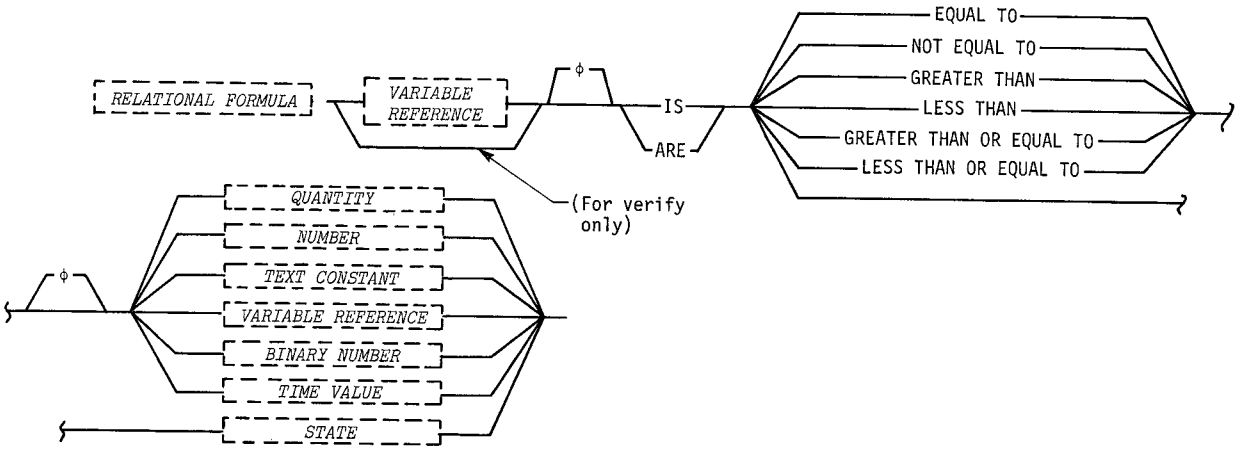
2.3.5 Formula Syntax



The syntactic unit "NUMERIC FORMULA" provides a syntactical structure for the expression of arithmetic calculations. The meaning of the symbols included in the syntax are:

- +, preceding the feedback loop, is unary positive;
- -, preceding the feedback loop, is negation;
- +, inside the feedback loop, is addition;
- -, inside the feedback loop, is subtraction;
- * is multiplication;
- / is division;
- ** is exponentiation;
- Parentheses enclose numeric formulas used within numeric formulas, where necessary, and also enclose quantities so as to delimit dimensional information to alleviate confusion of dimensional symbols and arithmetic symbols.

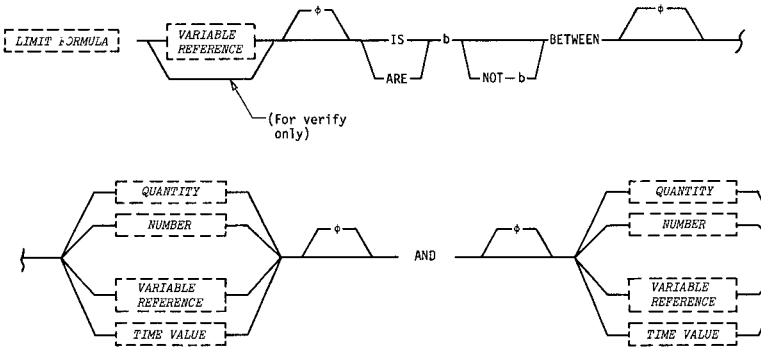
The use of incompatible variables and data constants will be flagged by the language processor as errors.



The syntactic unit "RELATIONAL FORMULA" provides a syntactical structure for the expression of relationships between variables or between variables and data constants. The use of incompatible variables and data constants will be flagged by the language processor as errors. If *VARIABLE REFERENCE* is *STATE*, the *VARIABLE REFERENCE* must be a Boolean variable. Relational formulas are used in the sequence control statements (defined later).

In the case of a *VARIABLE REFERENCE* that has a message data characteristic, only the relational phrases "IS EQUAL TO" or "IS NOT EQUAL TO" are allowable. This allows the comparison of a message type variable reference, input as a result of a read keyboard statement, with a precanned message variable or a text constant.

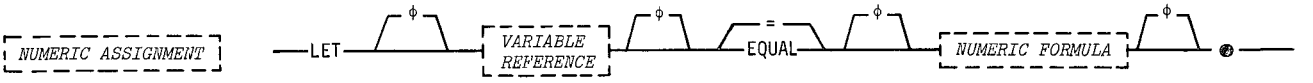
If the relational formula is used in a verification statement, then the first variable reference is omitted.



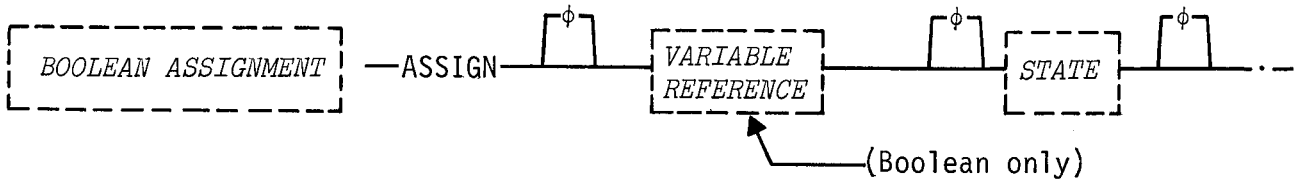
The syntactic unit "LIMIT FORMULA" provides a syntactical structure for the convenient expression of a relationship involving upper and lower limits on a variable reference. It is a compact version of a combination of relational formulas and follows the same rules with respect to incompatible variables and data constants.

If the limit formula is used in a verification statement, then the first variable reference is omitted.

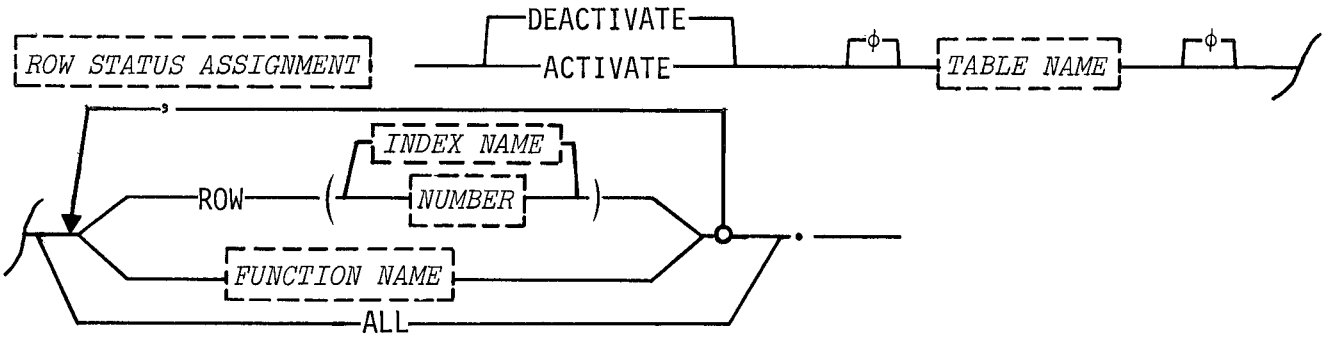
2.3.6 Assignment Statement Syntax



The syntactic unit "NUMERIC ASSIGNMENT" provides a syntactical structure for the assignment of values to a variable reference through the use of numeric formulas. The assignment of incompatible values to a variable reference will be flagged by the language processor as an error.

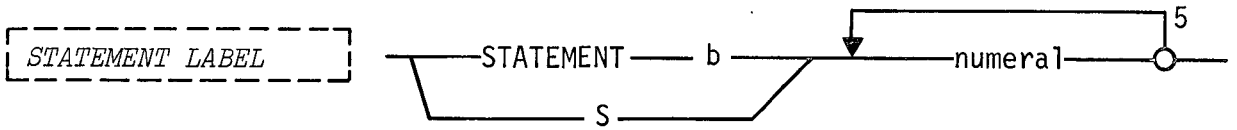


The syntactic unit "BOOLEAN ASSIGNMENT" provides a syntactical structure for the assignment of a state to a boolean variable reference. This is available for setting flags which are used for internal sequence control within a program.

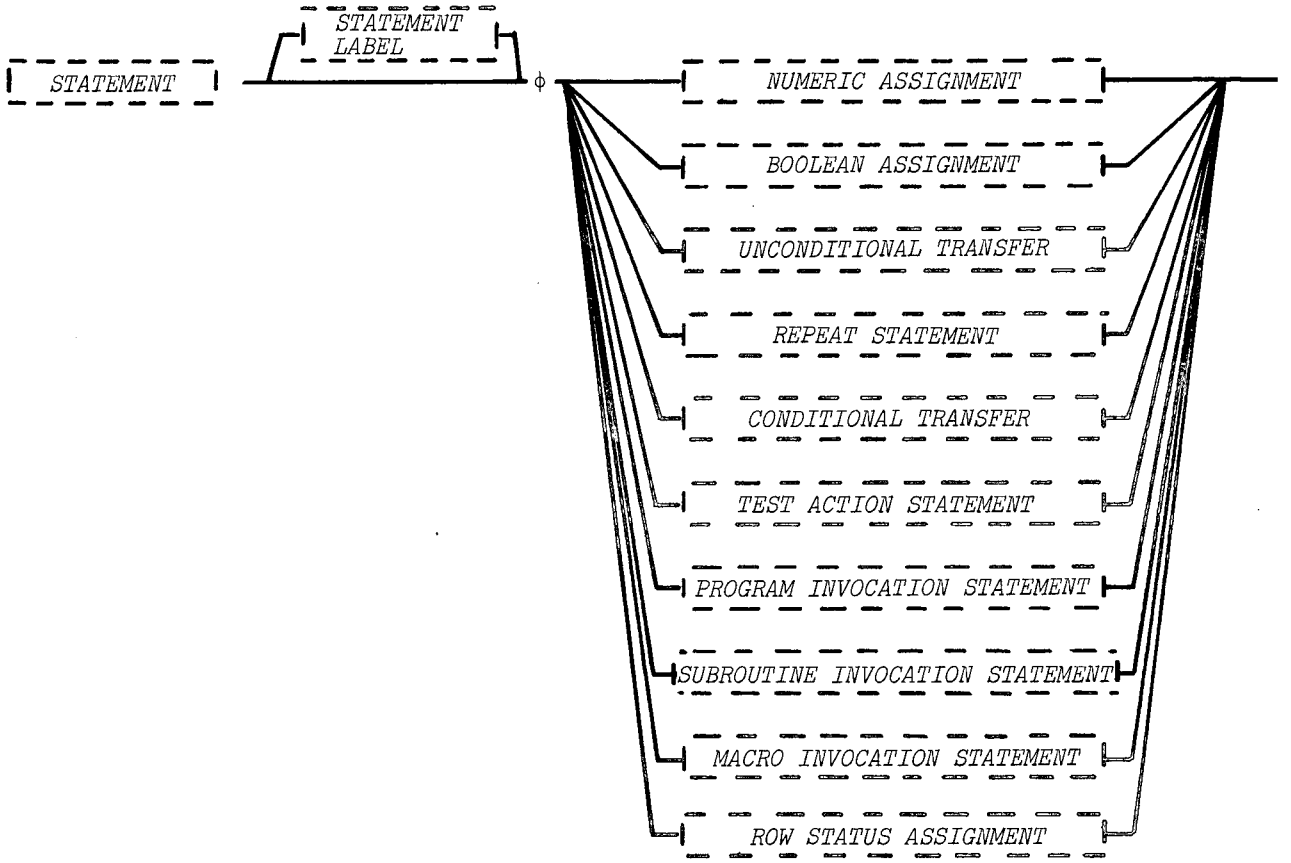


The syntactic unit "ROW STATUS ASSIGNMENT" provides a syntactical structure for the assignment of active or inactive status to individual rows of a table identified by the table name. No operations can be performed on those rows of a table that are inactive. The language processor takes into account the status of each row in determining whether to execute any required actions upon that row as indicated by other language statements.

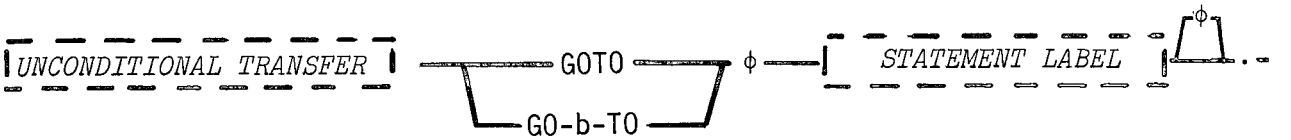
2.3.7 Sequence Control Syntax



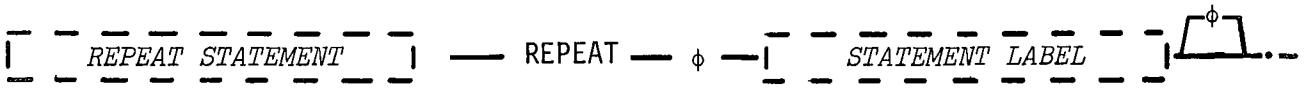
The syntactic unit "STATEMENT LABEL" is used to identify a statement (defined below) to which it is prefixed. The statement number is an arbitrarily chosen sequence of numbers, containing at least one and up to six individual numbers.



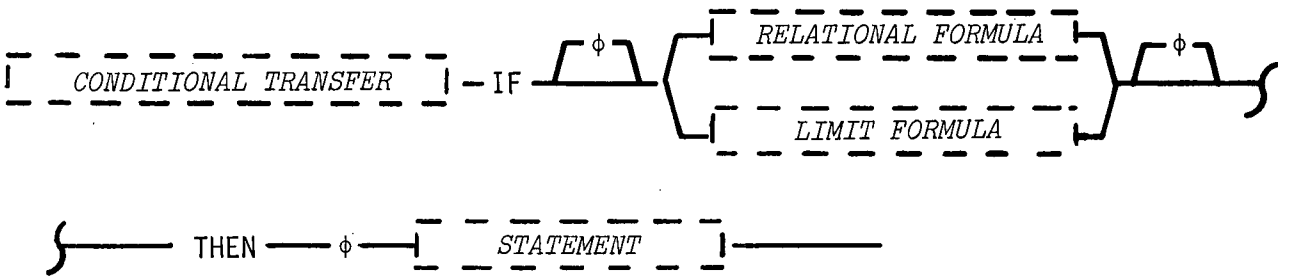
The syntactic unit "STATEMENT" is one of the possible statements illustrated. The numeric and Boolean assignment statements have been previously defined; the others are defined later.



The syntactic unit "UNCONDITIONAL TRANSFER" provides a syntactical structure for a transfer to a statement prefixed by a statement label.



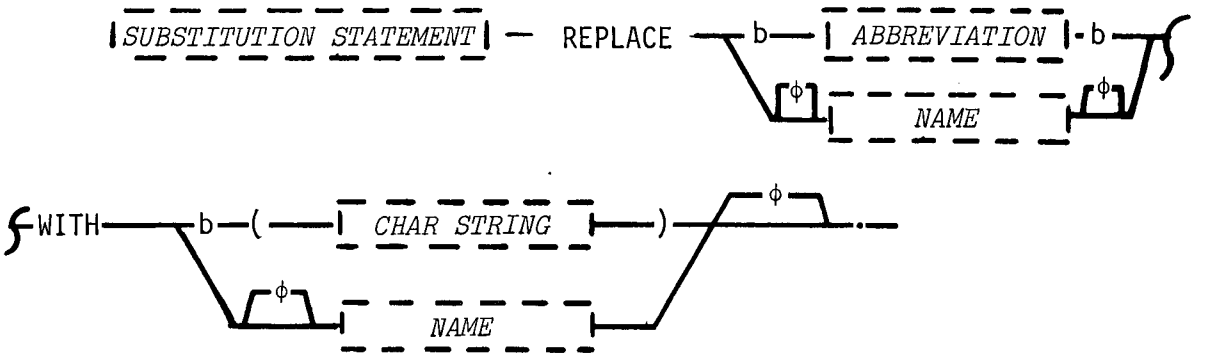
The syntactic unit "REPEAT STATEMENT" provides a syntactical structure for the single repetition of the single statement which is prefixed with the statement label indicated.



The syntactic unit "CONDITIONAL TRANSFER" provides a syntactical structure for the optional execution of a statement. The optional nature of the statement execution is provided by imbedding in the conditional transfer statement a relational formula or a limit formula. When the result of the evaluation of these imbedded syntactic units is "true," the statement following the "THEN" is executed. Otherwise, the statement is skipped and the next statement after the conditional transfer is executed. The statement following the "THEN" may often be an unconditional transfer.

If a table name and column name combination is used in the conditional transfer statement the relational or limit formula will be evaluated (on a row basis) for every individual item in the column as declared in the table declaration. Each individual formula evaluation must be true for the column evaluation to be true. A single failure of an evaluation constitutes a false evaluation for the column.

2.3.8 Definition Statement Syntax

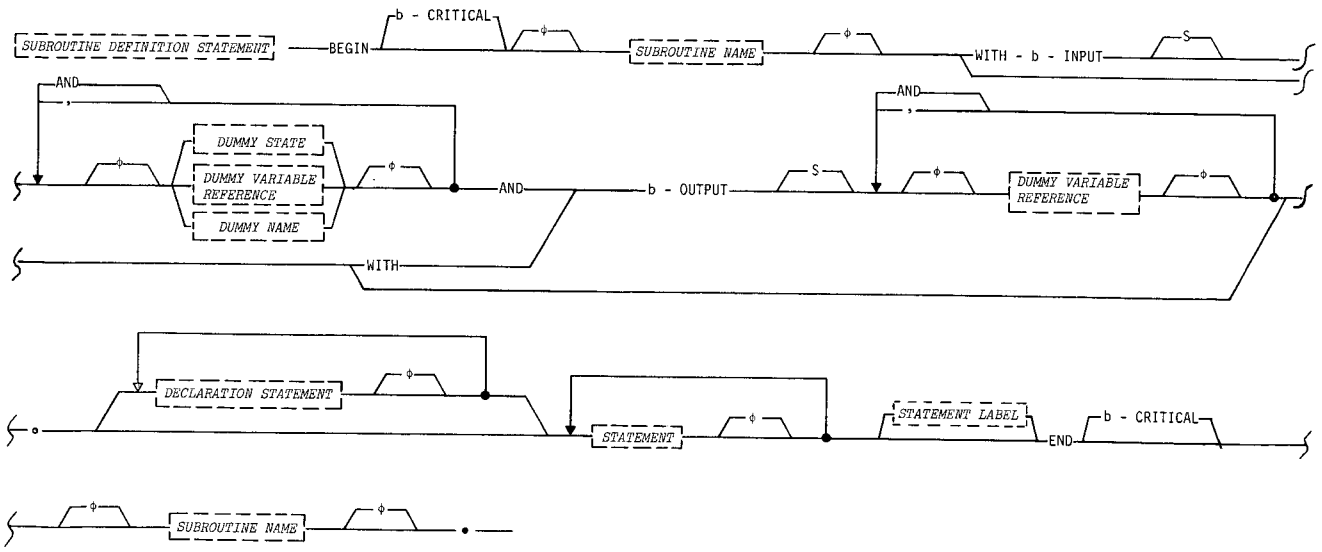


The syntactic unit "SUBSTITUTION STATEMENT" provides a syntactical structure for the substitution of character strings or names for other character strings or names in a program.

This structure provides the test writer with the capability of defining abbreviations which are then used in the source code of a program. When a source listing of the program is generated by the language processor, the abbreviations are removed and the full character strings are used. The character string may be a portion of a statement in the language or up to any number of statements. Substitution of the character string for the abbreviation is performed at processing time without modification of the character string from that defined in the abbreviation definition statement.

This structure also provides the test writer with the capability of substituting new names for names already in use within a program. This provides a simple method for modifying names used within a program to match those names which may have been changed in a dictionary data bank used by the program.

In either case, the abbreviation or name appearing in the source statements written by the test writer is identified on the left in the syntax diagram. The character string or name to be substituted into the source code by the language processor is identified on the right in the syntax diagram.

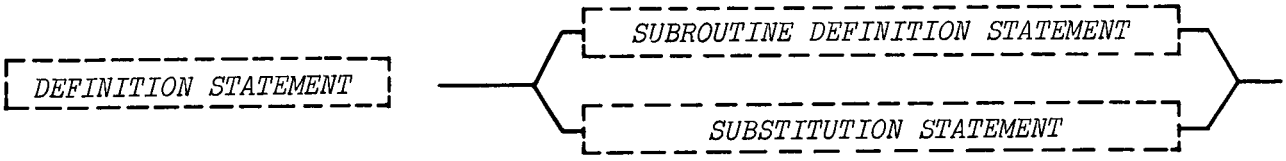


The syntactic unit "SUBROUTINE DEFINITION STATEMENT" provides a syntactical structure for the definition of subroutines. Two types of subroutines are identified; critical and noncritical. The term critical means that the actions being performed in the subroutine are time-dependent and as such, must not be interfered with. Interrupts are not allowed to break into the execution of such a subroutine. If the system in which the subroutine is being executed is running in a concurrent test execution mode, a critical subroutine causes suspension of the concurrent mode until it is complete.

Subroutines may be defined with inputs and outputs, with outputs alone, or with neither inputs or outputs. Inputs may be stated for test actions (defined later), variable references, or function names for test actions (defined later). These are identified in a subroutine definition by the appropriate dummy parameters. Outputs are variable references identified by the appropriate dummy parameters.

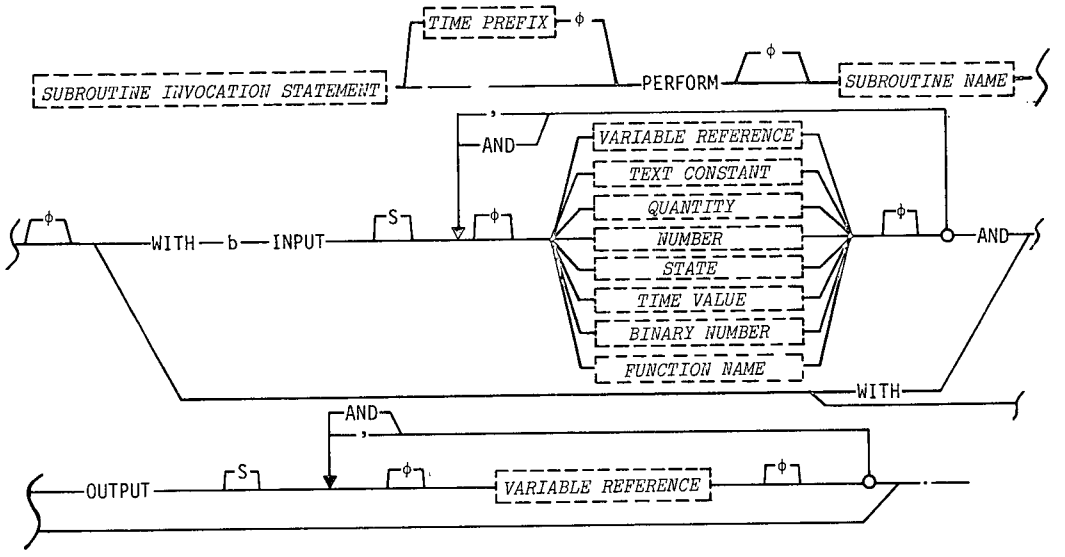
All dummy variable references require the corresponding declaration statements which establish the required data characteristics to be written within the subroutine definition.

The statement label on the end subroutine statement of the syntax provides a branch point for use when an exit from the subroutine is necessary before the ordinary sequence of execution is complete.

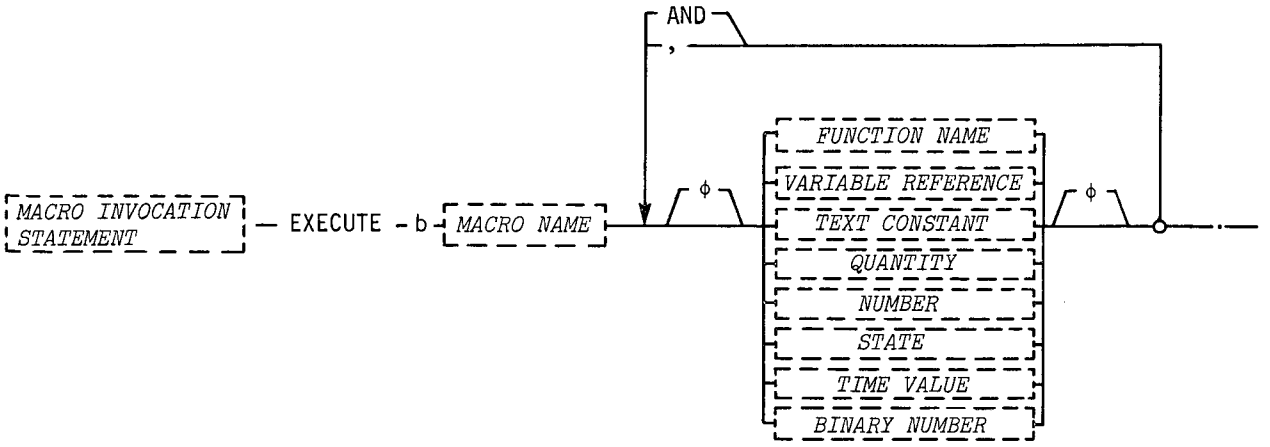


The syntactic unit "DEFINITION STATEMENT" is a subroutine definition statement or a substitution statement.

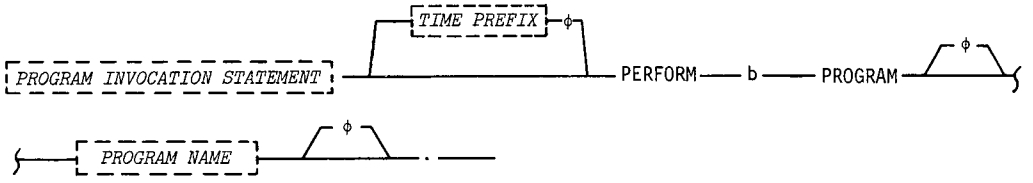
2.3.9 Invocation Statement Syntax



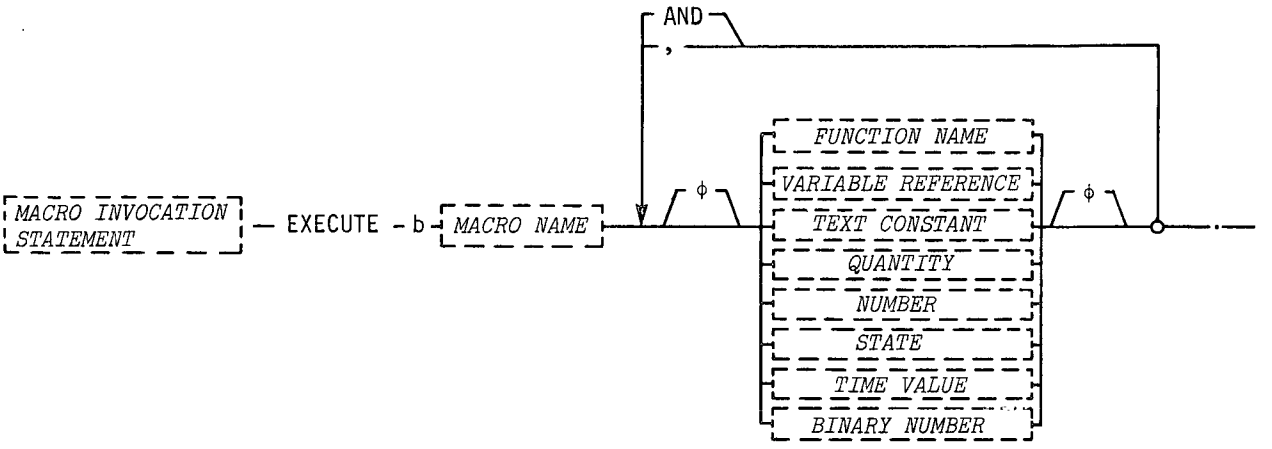
The syntactic unit "SUBROUTINE INVOCATION STATEMENT" provides a syntactical structure for the initiation of execution of a previously defined subroutine. The dummy state of the subroutine definition is replaced in the invocation by the syntactic unit *STATE*, as used in test action statements. The dummy name is replaced by a function name. All other syntactic units representing data in the invocation statement may be used to replace the dummy variable reference of the subroutine definition statement. These replacements must be consistent with the data characteristics declared for the dummy parameters. A time prefix may be attached to a subroutine invocation statement.



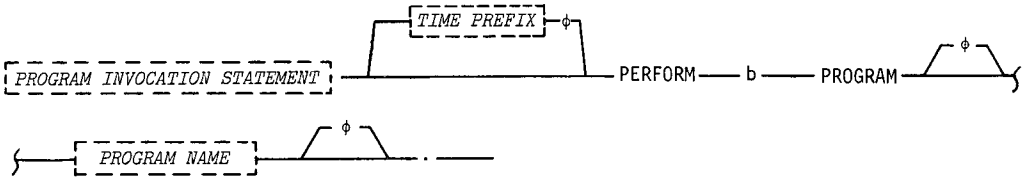
The syntactic unit "MACRO INVOCATION STATEMENT" provides a syntactical structure for the substitution of the macro definition (provided in a dictionary data bank, defined later) into the source code created by the test writer with appropriate substitutions for the dummy parameters in the macro definition. The resulting source listing provides the statements from the macro definition in place of the macro invocation statement.



The syntactic unit "PROGRAM INVOCATION STATEMENT" provides a syntactical structure for the initiation of a program by another program. A time prefix may be attached to a subroutine invocation statement.

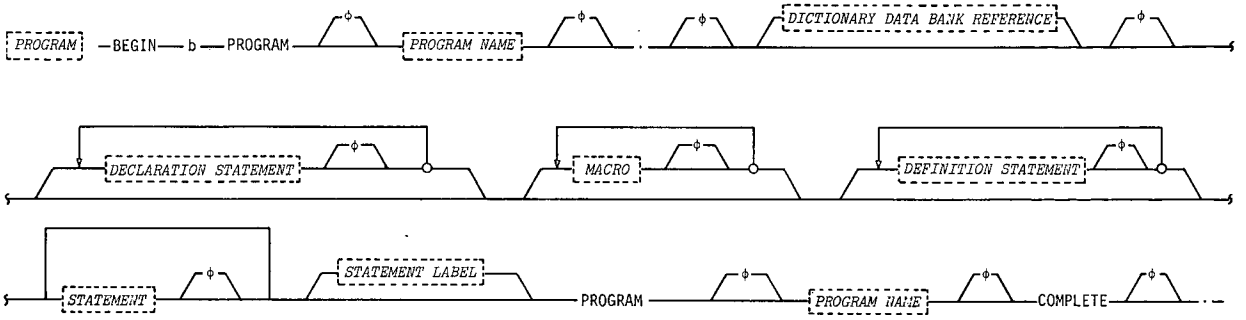


The syntactic unit "MACRO INVOCATION STATEMENT" provides a syntactical structure for the substitution of the macro definition (provided in a dictionary data bank, defined later) into the source code created by the test writer with appropriate substitutions for the dummy parameters in the macro definition. The resulting source listing provides the statements from the macro definition in place of the macro invocation statement.



The syntactic unit "PROGRAM INVOCATION STATEMENT" provides a syntactical structure for the initiation of a program by another program. A time prefix may be attached to a subroutine invocation statement.

2.3.10 Program Syntax



The syntactic unit "PROGRAM" provides the highest syntactical structure available in the language; that of a complete program.

The dictionary data bank reference (defined later) provides, via a selection of dictionary data banks, all information needed to define the functions of the line replaceable units and subsystems to be tested, channel addresses for telemetry data, and any data that is common for a number of separately processed programs. It also constitutes a library of subroutines and macros available for the use of the test writer.

The declaration statements provide the data declarations for all variables that are used only within the program being defined.

The definition statements provide the subroutines and macros that are used only within the program being defined. All abbreviations and substitutions used by the test writer for this particular program are also provided in the definition statements.

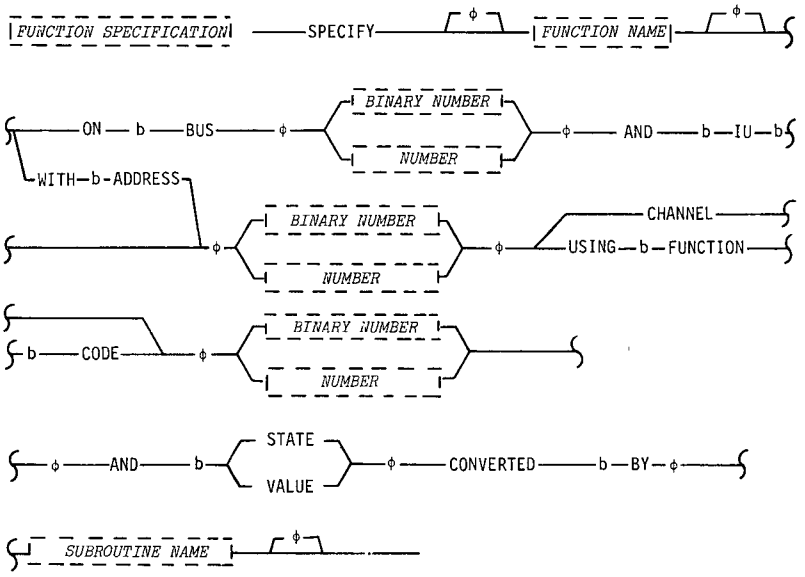
Finally, the statements that make up the main body of the program are included.

The statement label on the program complete statement of the syntax provides a branch point for use when an exit from the program is necessary before the ordinary sequence of execution is complete.

2.4 Dictionary Data Bank Syntax

The dictionary data bank provides the final link between the language and any specific test system. This is accomplished by providing, to the line replaceable unit (LRU) designers and the test equipment designers, the capability to declare the nouns required to test a unit and to define the action of the test system with respect to these nouns.

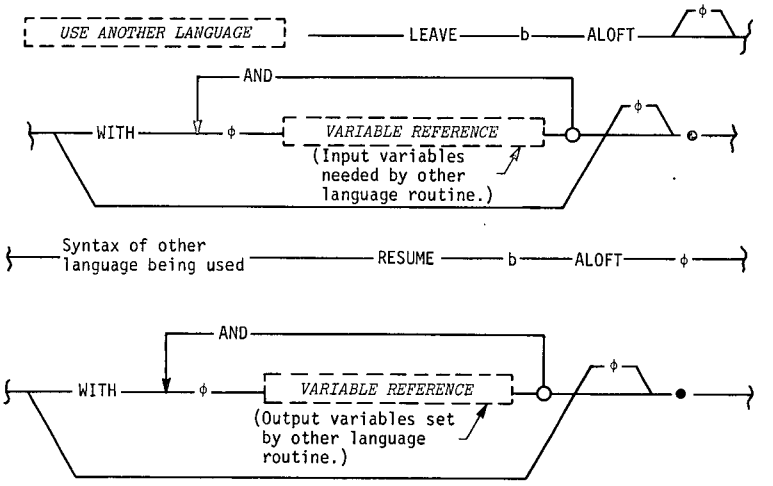
The dictionary data bank also provides for the definition of subroutines and macros, and the declaration of data variables that are common to a number of separately processed programs.



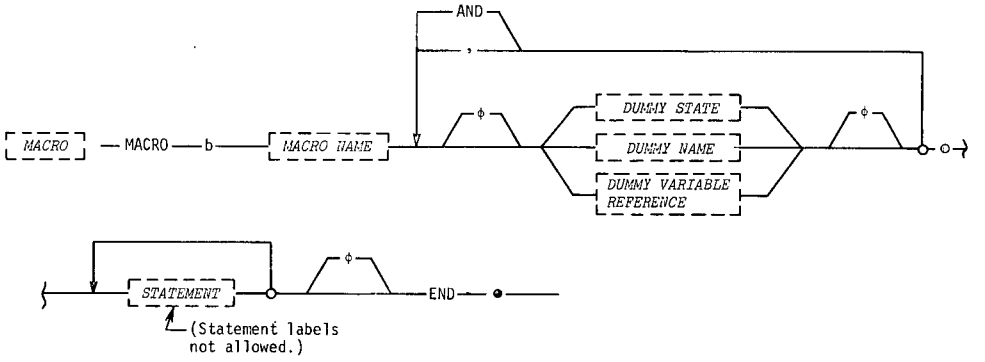
The syntactic unit "FUNCTION SPECIFICATION" provides a syntactical structure for the declaration of nouns that represent functions attached to a particular LRU. The LRU and its function is identified by a name, "FUNCTION NAME."

In the case of a shuttle-type data bus system the data bus address and interface unit address are provided as indicated by one option branch in the syntax. In the case of a telemetry system, an address is provided as indicated by the other option branch. In the case of a data bus system the syntax provides for the definition of the code that is sent on the data bus to effect the desired function. In the case of a telemetry system the channel address is defined. Finally, a subroutine is specified that provides the necessary code conversions for any inputs or outputs which result from the action of the function codes or for the data identified by the channel addresses.

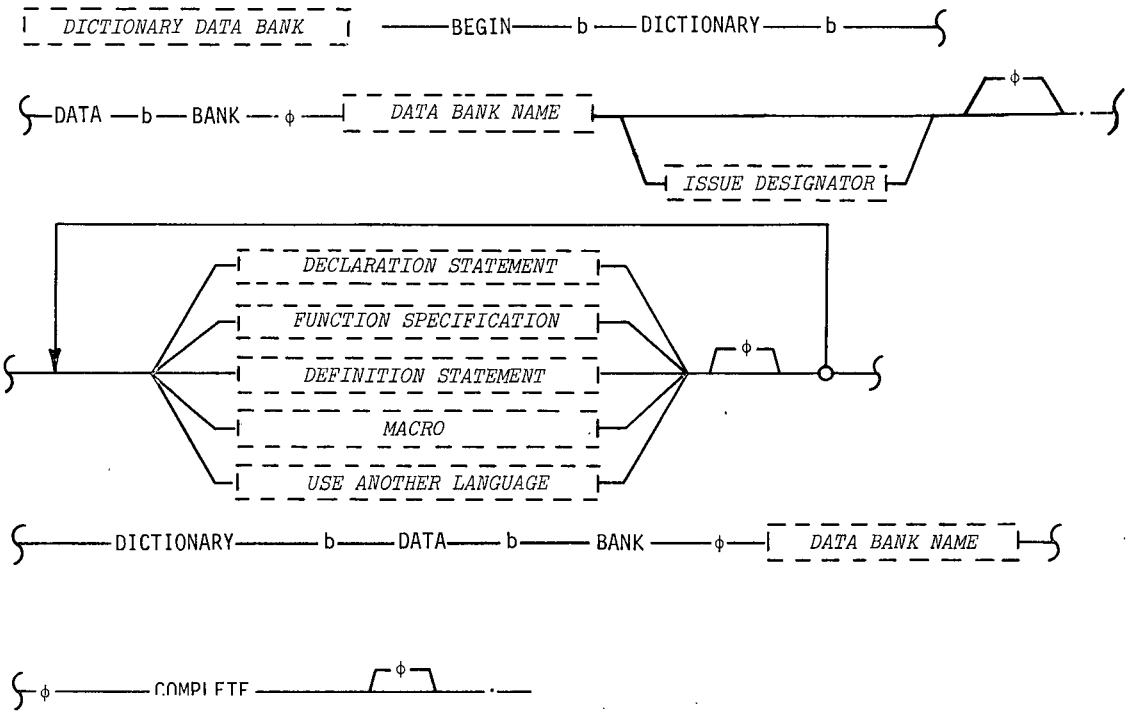
Commentary may be included in the function specification to identify such things as a function number generally attached to a function name. Ranges of values expected and other characteristics of data resulting from the action of the functions may be included in comments. These comments would be of assistance to the test writer as he writes tests that make use of function declarations.



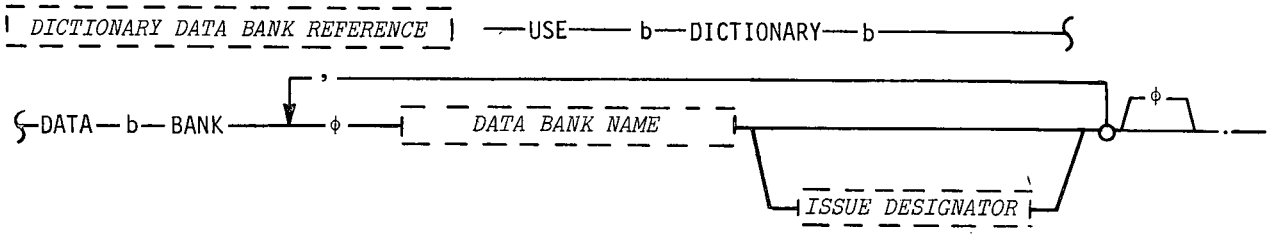
The syntactic unit "USE ANOTHER LANGUAGE" provides a syntactical structure for leaving the language, executing subroutines written in another language, and reentering the language. The variable references are provided to enable the passing of data back and forth between the languages. This capability would be used inside other subroutines defined in the dictionary data bank.



The syntactic unit "MACRO" provides a syntactical structure for the definition of new language functions based on language functions already available. The dummy parameters identified in the syntax are used in the statements that make up the macro definition. Statement labels are not allowed in the macro definition.



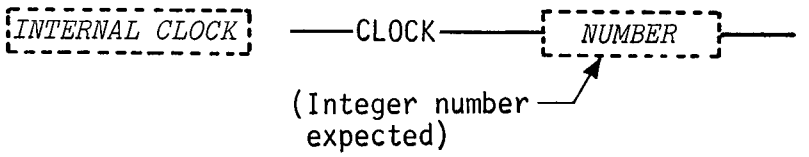
The syntactic unit "DICTIONARY DATA BANK" provides a syntactical structure for the definition of dictionary data banks. The dictionary data bank is made up of declaration statements, function specifications, definition statements, macros, and other language subroutines.



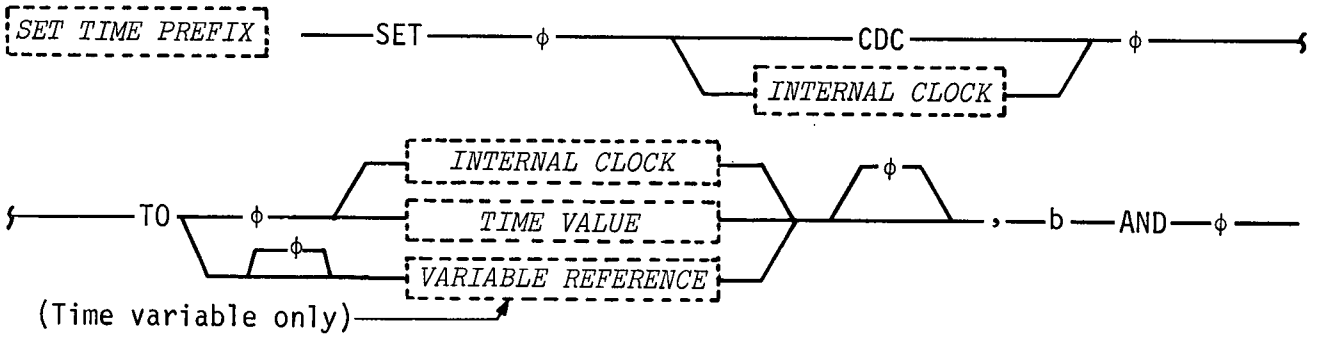
The syntactic unit "DICTIONARY DATA BANK REFERENCE" provides a syntactical structure that allows the test writer to identify those dictionary data banks that are required for a particular test.

2.5 Test Action Syntax

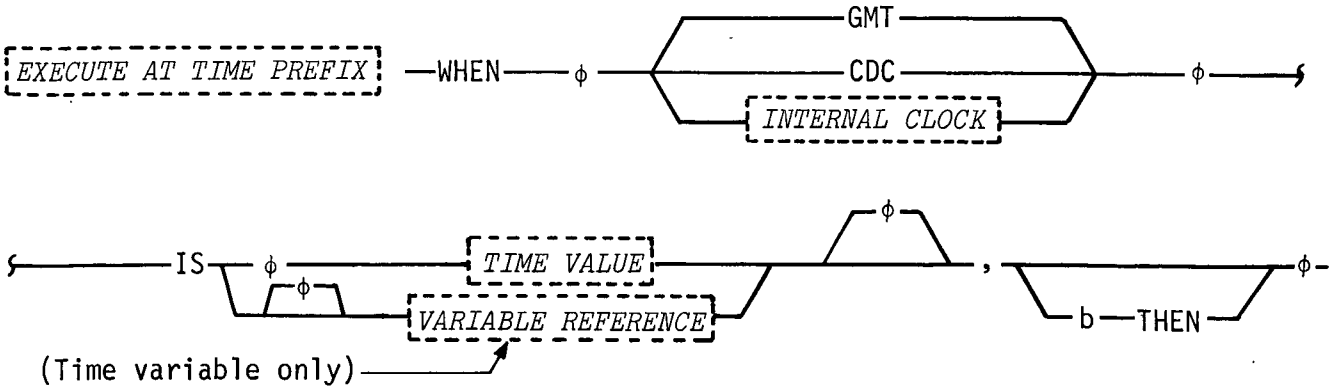
2.5.1 Clock and Time Controlled Action Syntax



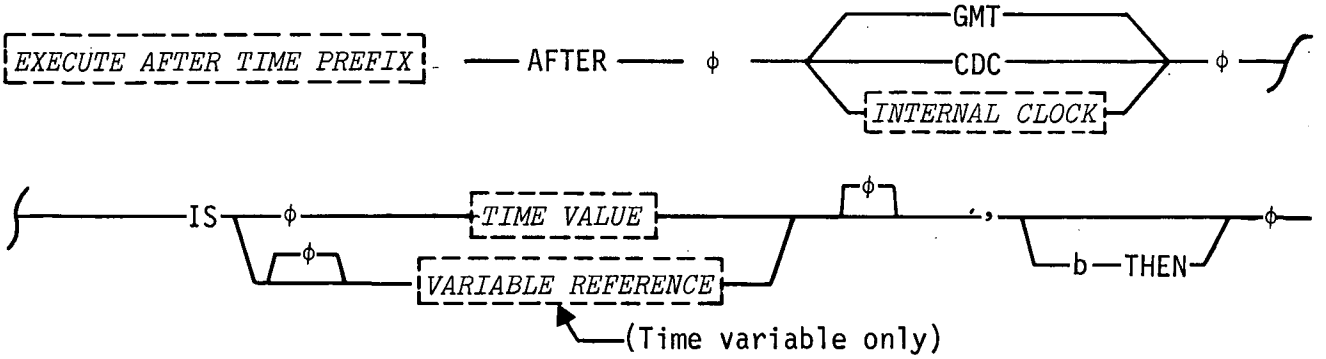
The syntactic unit "INTERNAL CLOCK" provides an identification for vehicle clocks, internal computer clocks, and software clocks which may be available in an executive system. An individual clock is identified by an integer number.



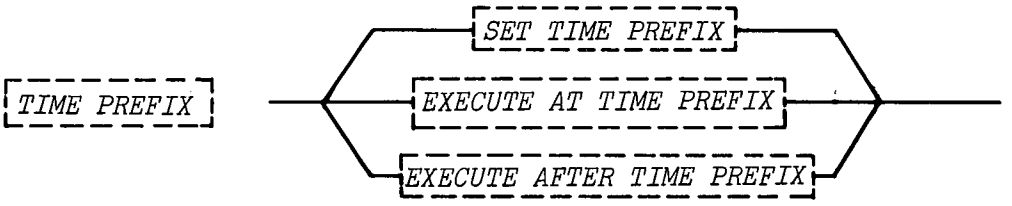
The syntactic unit "SET TIME PREFIX" provides a syntactical structure for the initialization of the countdown clock or an internal clock to a time or clock value at the time a statement to which the prefix is attached, is executed.



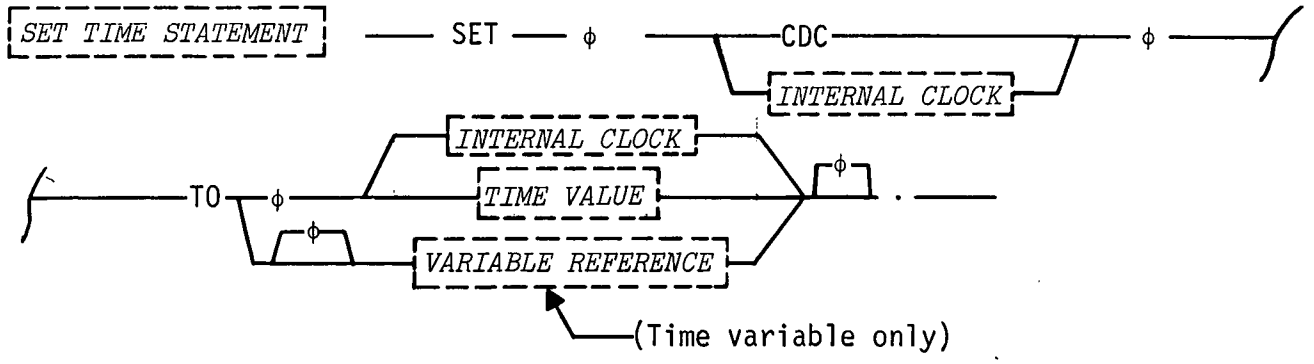
The syntactic unit "EXECUTE AT TIME PREFIX" provides a syntactical structure for the execution of a statement to which the prefix is attached, at the time identified in the prefix. Clocks identified are Greenwich Mean Time, countdown, or internal.



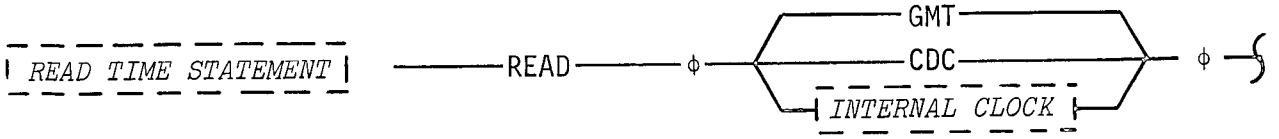
The syntactic unit "EXECUTE AFTER TIME PREFIX" provides a syntactical structure for the execution of a statement to which the prefix is attached, any time after the time identified in the prefix is reached.

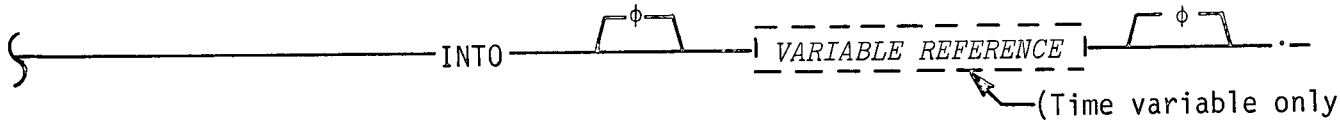


The syntactic unit "TIME PREFIX" is a set time prefix, an execute at time prefix, or an execute after time prefix.



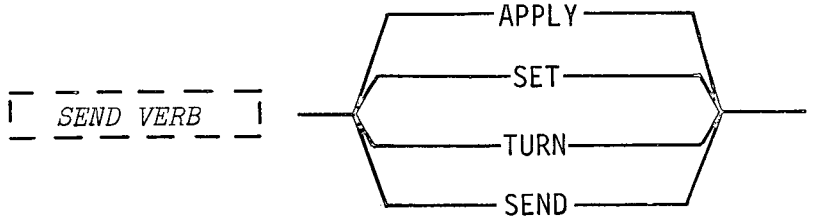
The syntactic unit "SET TIME STATEMENT" provides a syntactical structure for the initialization of the countdown clock or an internal clock to a particular time value, or another clock value.



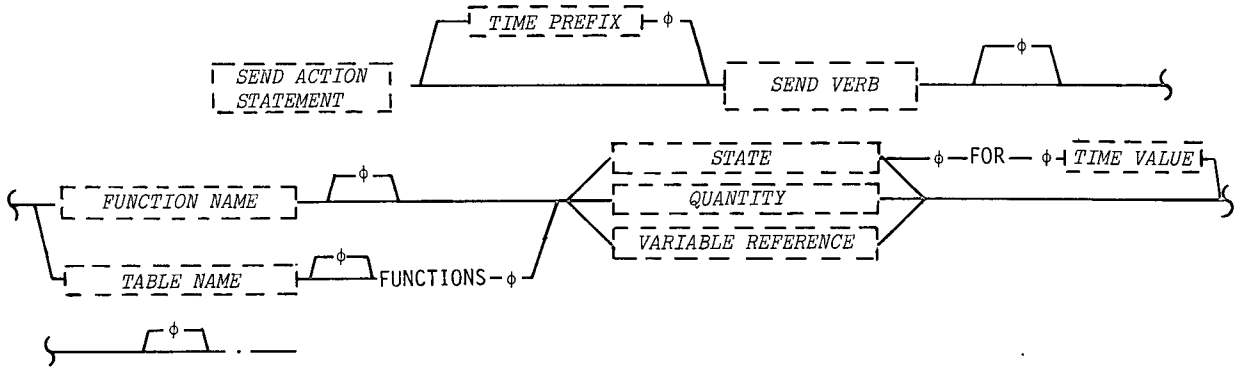


The syntactic unit "READ TIME STATEMENT" provides a syntactical structure for acquiring a time value, either Greenwich Mean Time, countdown time, or an internal clock time, and saving the value in a time type data variable.

2.5.2 Send Action Syntax



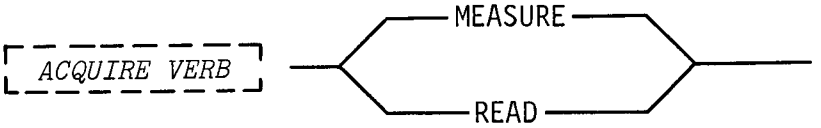
The syntactic unit "SEND VERB" provides terms for use in describing the send actions performed in send action statements, defined below.



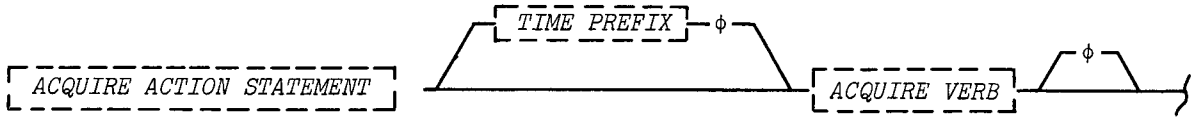
The syntactic unit "SEND ACTION STATEMENT" provides a syntactic structure for the performance of stimulus actions in a test. A time prefix may be attached to the send action statement. The function name is provided to the test writer from a dictionary data bank. The state of a discrete, a numeric quantity, or a variable reference identifying a numeric quantity to be sent to an LRU can be identified. In the case of a discrete state, a time limit for the application of that discrete state may be established. At the end of the time specified, the discrete state will be reversed.

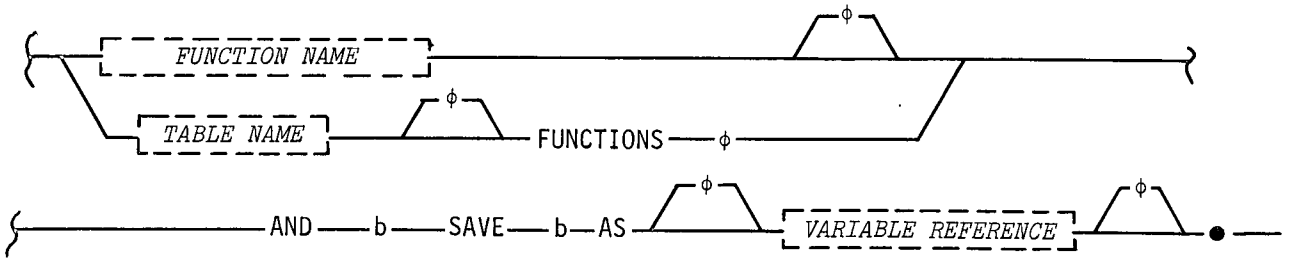
If a table name is identified instead of a function name, the function of each row of the table is sent with the appropriate values as identified by the state, quantity, or variable reference.

2.5.3 Acquire Action Syntax



The syntactic unit "ACQUIRE VERB" provides terms for use in describing the acquire actions performed in acquire action statements, defined below.

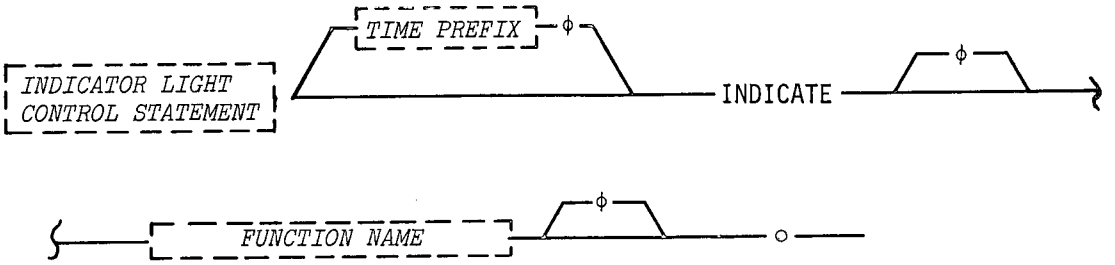




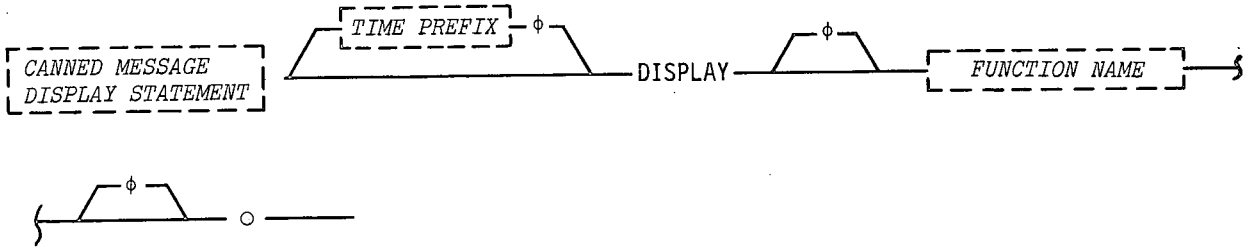
The syntactic unit "ACQUIRE ACTION STATEMENT" provides a syntactic structure for the performance of measurement actions in a test. A time prefix may be attached to the acquire action statement. The function name is provided to the test writer from a dictionary data bank. The information acquired by the action of this statement is retained, for later use in the test, in the variable reference identified.

If a table name is identified instead of a function name, the function of each row is acquired and saved using the appropriate variable reference identified for the retention of the resulting data.

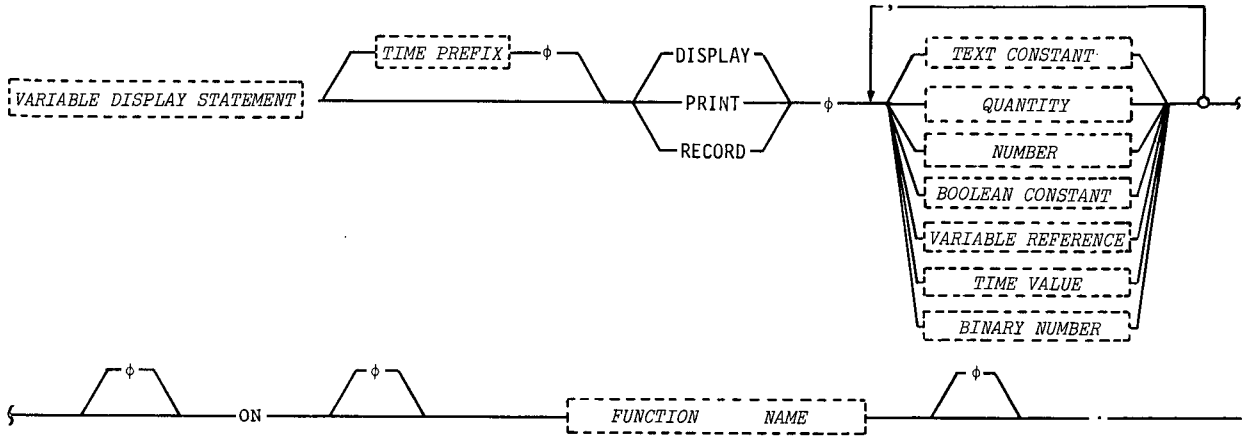
2.5.4 Display Action Syntax



The syntactic unit "INDICATOR LIGHT CONTROL STATEMENT" provides a syntactical structure for the control of indicator lights on a panel available to the test system operator in the shuttle cockpit. The lights are on/off type with possible color variations. Information concerning the light, its functions, and its location is provided by a dictionary data bank. A time prefix may be attached to an indicator light control statement.

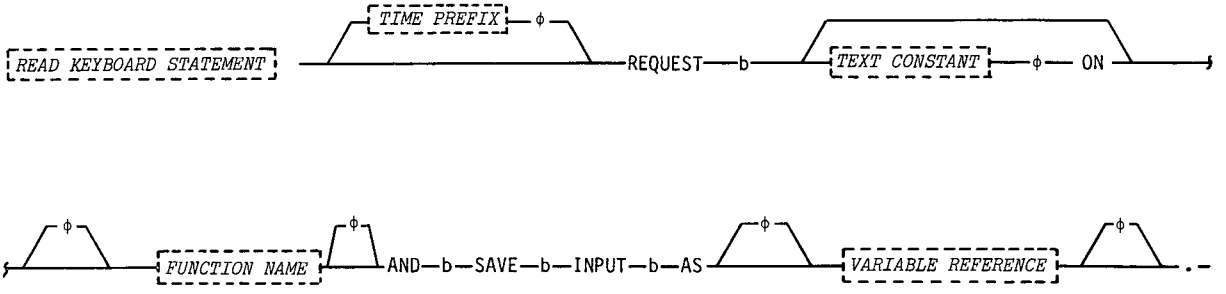


The syntactic unit "CANNED MESSAGE DISPLAY STATEMENT" provides a syntactical structure for the output of nonvariable messages. An example of such a message would be the display of information contained on a microfilm frame. Information concerning the address of the message and the output device used is provided by a dictionary data bank. A time prefix may be attached to a canned message display statement.



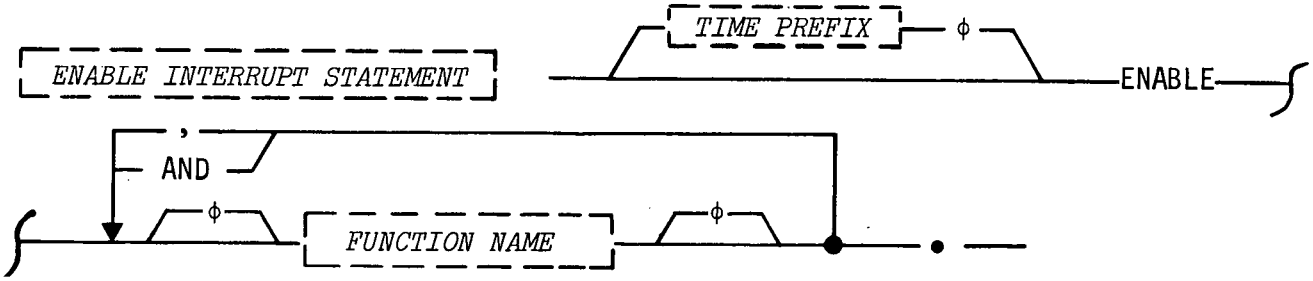
The syntactic unit "VARIABLE DISPLAY STATEMENT" provides a syntactical structure for formatting and displaying messages containing quantities whose values are dependent on run time conditions. This information may be displayed on a CRT alphanumeric light display, magnetic tape, or a printer. Full formatting capability is provided by the feedback loop in the syntax. Information concerning the address of the display device and its functions is provided by a dictionary data bank. A time prefix may be attached to the variable display statement.

The option "DISPLAY" permits the display of information on either the alphanumeric light display or CRT. The option "PRINT" permits the display of information on a printer. The option "RECORD" permits the identification and recording of data on the maintenance recorder or flight recorder of the Space Shuttle. The function name provides the device function (i.e., line number in the case of a CRT) and the device identifier. The appropriate function names must be used with the options described above.

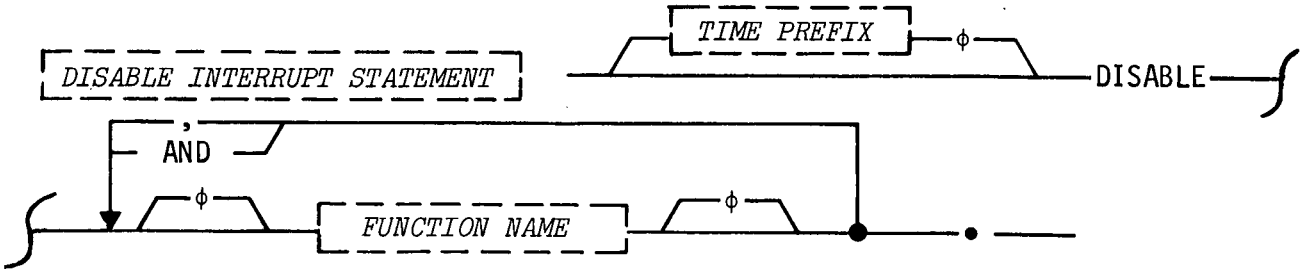


The syntactic unit "READ KEYBOARD STATEMENT" provides a syntactical structure for requesting an action from a test operator and saving the results of that action for later use. One option provides a canned message for display to the operator in a manner similar to the action of a canned message display statement. The other option allows a text message to be displayed to the operator in a manner similar to the action of the variable display statement. Following the display of the message requesting some action on the part of the test operator, an input message will be accepted from the keyboard and saved in the variable reference identified. A time prefix may be attached to a read keyboard statement.

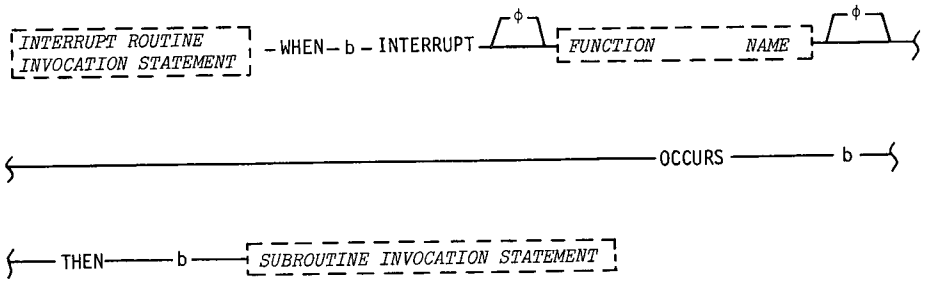
2.5.5 Interrupt Action Syntax



The syntactic unit "ENABLE.INTERRUPT STATEMENT" provides a syntactical structure for the enabling of an interrupt identified by the function name in a dictionary data bank. Any number of these interrupts may be enabled in a single statement. A time prefix may be attached to an enable interrupt statement.

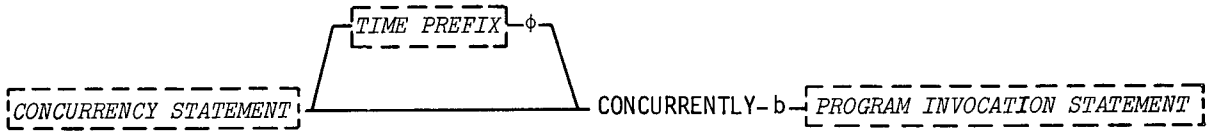


The syntactic unit "DISABLE INTERRUPT STATEMENT" provides a syntactical structure for the disabling of an interrupt identified by the function name in a dictionary data bank. Any number of these interrupts may be disabled in a single statement. A time prefix may be attached to a disable interrupt statement.

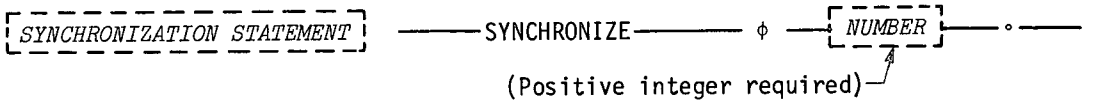


The syntactic unit "INTERRUPT ROUTINE INVOCATION STATEMENT" provides a syntactical structure for identifying a subroutine to be executed upon the occurrence of an interrupt. The interrupt is identified by a function name in a dictionary data bank. The subroutine is identified in a subroutine invocation statement.

2.5.6 Concurrent Testing Syntax

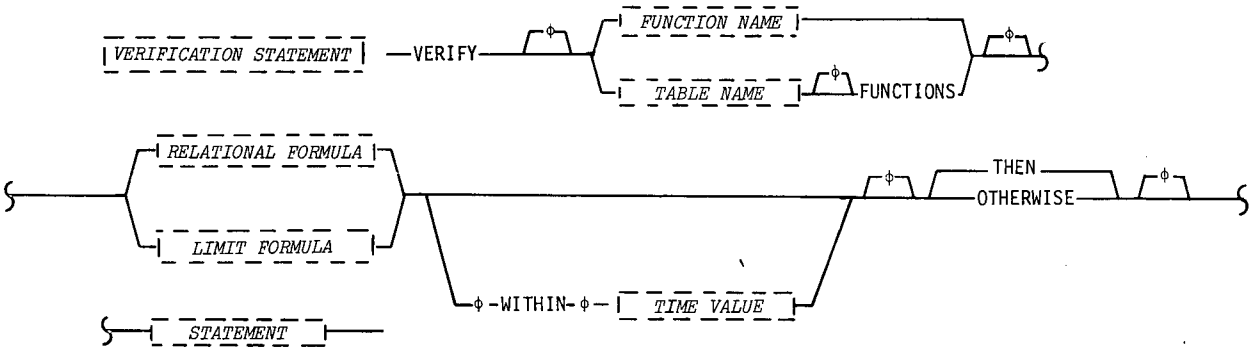


The syntactic unit "CONCURRENCY STATEMENT" provides a syntactical structure for calling up, for concurrent execution, a program identified in the program invocation statement. A time prefix may be attached to the concurrency statement.



The syntactic unit "SYNCHRONIZATION STATEMENT" provides a syntactical structure to identify synchronization points in a test. The positive integer number identifies each particular synchronization point. Identical synchronization points in two or more concurrently executing tests will cause the tests to be synchronized at that point.

2.5.7 Compound Action Statement Syntax



The syntactic unit "VERIFICATION STATEMENT" provides a syntactical structure for reading or measuring the LRU function identified by the function name or the LRU functions attached to a table, and then comparing the resulting values with other data via relational or limit formulas. If the "OTHERWISE" option is selected and the result of evaluation of the formula is false the statement following the "OTHERWISE" is executed. If the result of the evaluation of the formula is true the statement following the "OTHERWISE" is not executed.

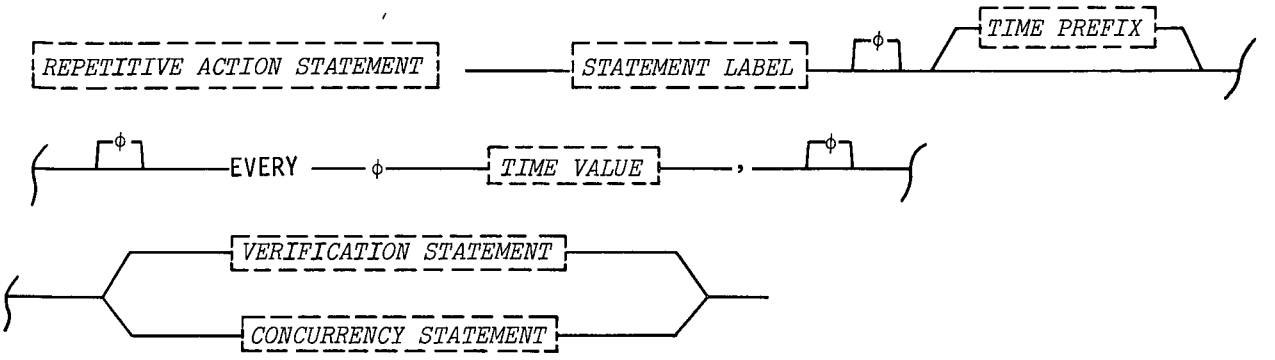
If the "THEN" option is selected and the result of the evaluation of the formula is true the statement following the "THEN" is executed. If the result of the evaluation of the formula is false the statement following the "THEN" is not executed.

In the case of the table name option, the relational or limit formula will be evaluated on a row basis for every individual item in the column as declared in the table declaration. Each individual formula evaluation must be true for the column evaluation to be true. A single failure of an evaluation constitutes a false evaluation for the column.

In any case, the value or values read as a result of execution of the verification statement are not retained.

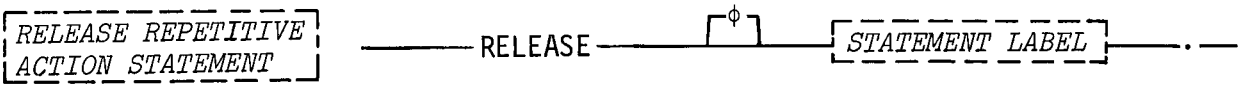
If the option "WITHIN TIME VALUE" is chosen, the verification statement means hold until the verification required occurs or the time value indicated is exceeded. If verification does not occur and the time value is exceeded the statement following the "OTHERWISE" will be executed.

2.5.8 Repetitive Action Syntax



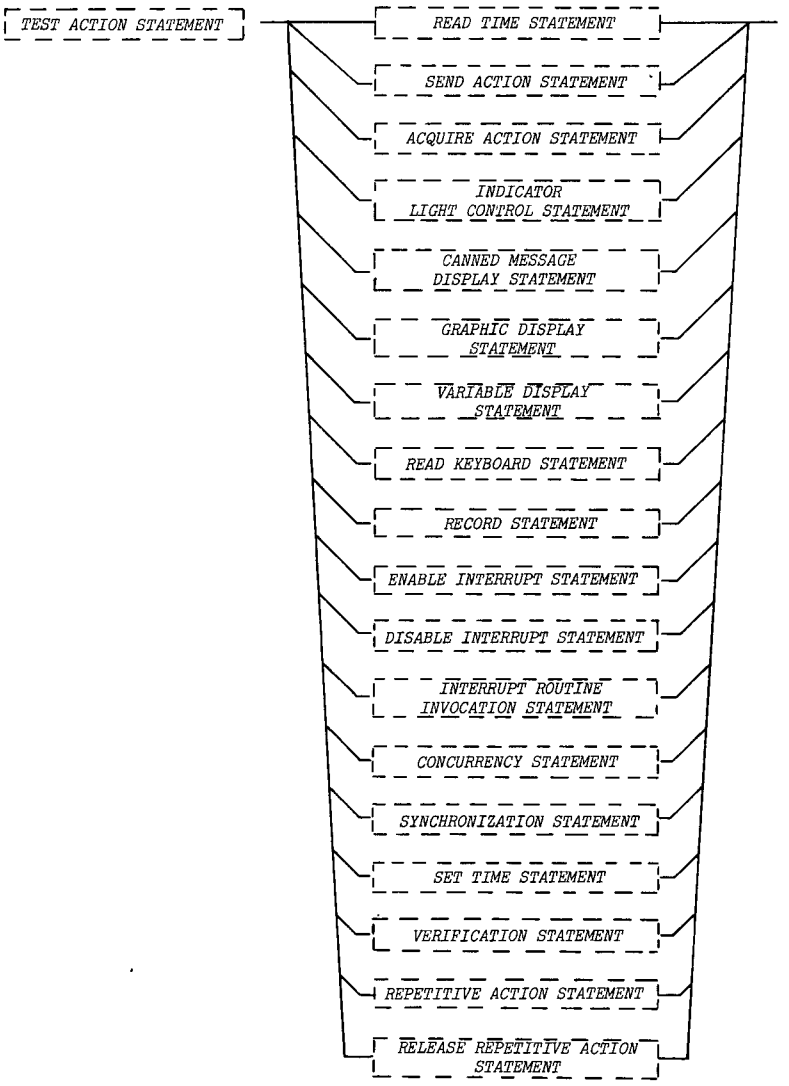
The syntactic unit "REPETITIVE ACTION STATEMENT" provides a syntactical structure for the definition of a repetitive action. A statement label must be attached to identify this particular statement for use in a release statement (defined later). The time value indicates how often to repeat the statement called for. Two types of statements may be repeated, which results in two different actions.

A repetition of a verification statement indicates a repetitive execution of the statement in line with the program in which the repetitive action statement resides. A repetition of a concurrency statement indicates a repetitive execution of a program, identified in the concurrency statement, in parallel with the program containing the repetitive action statement.



The syntactic unit "RELEASE REPETITIVE ACTION STATEMENT" provides a syntactical structure for the release from execution status of a verification statement or concurrently executing program identified for repetitive execution via a repetitive action statement.

2.5.9 Test Action Statements



The syntactic unit "TEST ACTION STATEMENT" is one of the total possible test action statements in the language.